EVOLUTIONARY COMPUTATION

Th. Bäck, Leiden Institute of Advanced Computer Science, Leiden University, NL and divis digital solutions GmbH, Dortmund, D.

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

Contents

Glossary

(μ, λ)-Strategy: See comma strategy.

$(\mu + \lambda)$-Strategy: See plus strategy.

Adaptation: Denotes the general advantage in ecological or physiological efficiency of an individual in contrast to other members of the population, and it also denotes the process of attaining this state.

Building Block: In the classical theory of genetic algorithms, short, low-order, above-average schemata are called building blocks. According to the schema theorem, such schemata in a population receive an exponentially increasing number of trials in the following generations, such that promising regions of the search space are sampled with an exponentially increasing number of representatives in the population.

Classifier Systems: Dynamic, rule-based systems capable of learning by examples and induction. Classifier systems evolve a population of production rules (in the so-called Michigan approach, where an individual corresponds to a single rule) or a population of production rule bases (in the so-called Pittsburgh approach, where an individual represents a complete rule base) by means of an evolutionary algorithm (typically a genetic algorithm). The rules are often encoded by a ternary alphabet, which contains a "don't care" symbol facilitating a generalization capability of condition or action parts of a rule, thus allowing for an inductive learning of concepts. In the Michigan approach, the rule fitness (its strength) is incrementally updated at each generation by the "bucket brigade" credit assignment algorithm based on the reward the system obtains from the environment, while in the Pittsburgh approach the fitness of a complete rule base can be calculated by testing the behavior of the individual within its environment.

Comma Strategy: The notation (μ, λ)-strategy describes a selection method introduced in evolution strategies and indicates that a parent population of μ individuals generates $\lambda > \mu$ offspring and the best out of these λ offspring are deterministically selected as parents of the next generation.

Computational Intelligence: The field of computational intelligence is currently seen to include subsymbolic approaches to artificial intelligence, such as neural networks, fuzzy systems, and evolutionary computation, which are gleaned from the model of information processing in natural systems. Following a commonly accepted characterization, a system is computationally intelligent if it deals only with numerical data, does not use knowledge in the classical expert system sense, and exhibits computational adaptivity, fault tolerance, and speed and error rates approaching human performance.

Convergence Reliability: Informally, the convergence reliability of an evolutionary algorithm means its capability to yield reasonably good solutions in case of highly

multimodal topologies of the objective function. Mathematically, this is closely related to the property of global convergence with probability one, which states that given infinite running time, the algorithm finds a global optimum point with probability one. From a theoretical point of view, this is an important property to justify the feasibility of evolutionary algorithms as global optimization methods.

Convergence Velocity: In the theory of evolutionary algorithms, the convergence velocity is defined either as the expectation of the change of the distance towards the optimum between two subsequent generations, or as the expectation of the change of the objective function value between two subsequent generations. Typically, the best individual of a population is used to define the convergence velocity.

Crossover: A process of information exchange of genetic material that occurs between adjacent chromatids during meiosis.

Defining Length: The defining length of a schema in genetic algorithm theory is the maximum distance between specified positions within the schema. The larger the defining length of a schema, the higher becomes its disruption probability by crossover.

Diffusion Model: The diffusion model denotes a massively parallel implementation of evolutionary algorithms, where each individual is realized as a single process being connected to neighboring individuals, such that a spatial individual structure is assumed. Recombination and selection are restricted to the neighborhood of an individual, such that information is locally preserved and spreads only slowly over the population.

Discrete Recombination: A recombination operator introduced in evolution strategies. Discrete recombination works on two vectors of object variables by performing an exchange of the corresponding object variables with probability one half (other settings of the exchange probability are in principle possible). Cf. uniform crossover.

DNA: Deoxyribonucleic acid, a double stranded macromolecule of helical structure (comparable to a spiral staircase). Both single strands are linear, unbranched nucleic acid molecules build up from alternating deoxyribose (sugar) and phosphate molecules. Each deoxyribose part is coupled to a nucleotide base, which is responsible for establishing the connection to the other strand of the DNA. The four nucleotide bases Adenine (A), Thymine (T), Cytosine (C) and Guanine (G) are the alphabet of the genetic information. The sequences of these bases in the DNA molecule determines the building plan of any organism.

Elitism: Elitism is a feature of some evolutionary algorithms ensuring that the maximum objective function value within a population can never reduce from one generation to the next. This can be assured by simply copying the best individual of a population to the next generation, if none of the selected offspring constitutes an improvement of the best value.

Evolutionary Algorithm: See evolutionary computation.

Evolutionary Computation: Encompasses methods of simulating evolution on a computer. The field is presently seen as including research in genetic algorithms, evolution strategies, evolutionary programming, genetic programming, classifier systems, and related techniques. The three classical representatives of evolutionary computation, namely genetic algorithms, evolution strategies, and evolutionary programming, are often also referred to as evolutionary algorithms.

Evolution Strategy: An evolutionary algorithm developed by I. Rechenberg and H.-P. Schwefel at the Technical University of Berlin, Germany, in the 1960s. The evolution strategy typically employs real-valued parameters, though it has also been used for discrete problems. Its basic features are the distinction between a parent population (of size μ) and an offspring population (of size $\lambda \geq \mu$), the explicit emphasis on normally distributed mutations, the utilization of different forms of recombination, and the incorporation of the self-adaptation principle for strategy parameters, i.e., those parameters that determine the mutation probability density function are evolved on-line, by the same principles which are used to evolve the object variables.

Evolutionary Programming: An evolutionary algorithm developed by L. J. Fogel at San Diego CA in the 1960s and further refined by D. B. Fogel and others in the 1990s. Evolutionary programming was originally developed as a method to evolve finite-state-machines for solving time series prediction tasks and was later extended to parameter optimization problems. The basic features of evolutionary programming include the fact that typically no recombination is used and the algorithm relies on mutation as the basic search operator. Selection is a stochastic tournament selection that determines μ individuals to survive out of the μ parents and the μ (or other number of) offspring generated by mutation. Evolutionary programming also uses the self-adaptation principle to evolve strategy parameters on-line during the search (cf. evolution strategies).

Fitness: The propensity of an individual to survive and reproduce in a particular environment. In evolutionary algorithms, the fitness value of an individual is closely related (and sometimes identical) to the objective function value of the solution represented by the individual, but especially in genetic algorithms using proportional selection a scaling function is typically necessary to map objective function values to positive values such that the best performing individual receives maximum fitness.

Generation Gap: The generation gap characterizes the percentage of the population to be replaced during each generation. The remainder of the population is chosen (at random) to survive intact. The generation gap allows for gradually shifting from the generation-based working scheme towards the extreme of just generating one new individual per "generation," the so-called steady-state selection algorithm.

Genetic Algorithm: An evolutionary algorithm developed by J. H. Holland at Ann Arbor MI in the 1960s. Originally, the genetic algorithm or adaptive plan was designed

as a formal system for adaptation rather than an optimization system. Its basic features are the strong emphasis on recombination (crossover), use of a probabilistic selection operator (proportional selection), and the interpretation of mutation as a background operator, playing a minor role for the algorithm. While the original form of genetic algorithms (the canonical genetic algorithm) represents solutions by binary strings, a number of variants including real-coded genetic algorithms and order-based genetic algorithms have also been developed to make the algorithm applicable to other than binary search spaces.

Genetic Programming: Derived from genetic algorithms, the genetic programming paradigm characterizes a class of evolutionary algorithms aiming at the automatic generation of computer programs. To achieve this, each individual of a population represents a complete computer program in a suitable programming language. Most commonly, symbolic expressions representing parse trees in (a subset of) the LISP language are used to represent these programs, but also other representations (including binary representation) and other programming languages (including machine code) are successfully employed.

Genotype: The sum of inherited characters maintained within the entire reproducing population. Often also the genetic constitution underlying a single trait or set of traits.

Global Optimization: Given a function $f : M \to I\!R$, the problem of determining a point $\vec{x}^* \in M$ such that $f(\vec{x}^*)$ is minimal (i.e., $f(\vec{x}^*) \leq f(\vec{x})\ \forall \vec{x} \in M$) is called the global optimization problem.

Global Recombination: In evolution strategies, recombination operators are sometimes used which potentially might take all individuals of a population into account for the creation of an offspring individual. Such recombination operators are called global recombination (i.e., global discrete recombination or global intermediate recombination).

Gray Code: A binary code for integer values which ensures that adjacent integers are encoded by binary strings with Hamming distance one. Gray codes play an important role in the application of canonical genetic algorithms to parameter optimization problems, because both theoretical and empirical findings indicate that they are preferable if a binary coding of real values is desired.

Implicit Parallelism: The implicit parallelism result of the classical schema theory of genetic algorithms claims that, given a population size of μ individuals, the number of schemata processed at each generation is of the order μ^3.

Individual: A single member of a population. In evolutionary algorithms, an individual contains a chromosome or genome, that usually contains at least a representation of a possible solution to the problem being tackled, i.e., a single point in the search space. Other information such as certain strategy parameters and the individual's fitness value are usually also stored in each individual.

Intermediate Recombination: A recombination operator introduced in evolution strategies. Intermediate recombination performs an averaging operation on the components of the two parent vectors.

Meta-Evolution: The problem of finding optimal settings of the exogeneous parameters of an evolutionary algorithm can itself be interpreted as an optimization problem. Consequently, the attempt has been made to use an evolutionary algorithm on the higher level to evolve optimal strategy parameter settings for evolutionary algorithms, thus hopefully finding a best-performing parameter set that can be used for a variety of objective functions. The corresponding technique is often called a meta-evolutionary algorithm. An alternative approach involves the self-adaptation of strategy parameters by evolutionary learning.

Migration: The transfer of an individual from one subpopulation to another.

Migration Model: The migration model (often also referred to as the island model) is one of the basic models of parallelism exploited by evolutionary algorithm implementations. The population is no longer panmictic, but distributed into several independent subpopulations (so-called demes), which co-exist (typically one different processors, with one subpopulation per processor) and may mutually exchange information by inter-deme migration. Each of the sub-populations corresponds to a conventional (i.e., sequential) evolutionary algorithm. Since selection takes place only locally inside a population, every deme is able to concentrate on different promising regions of the search space, such that the global search capabilities of migration models often exceed those of panmictic populations. The fundamental parameters introduced by the migration principle are the exchange frequency of information, the number of individuals to exchange, the selection strategy for the emigrants, and the replacement strategy for the immigrants.

Multi-Point Crossover: A crossover operator in genetic algorithms which uses a predefined number of uniformly distributed crossover points and exchanges alternating segments between pairs of crossover points between the parent individuals. Cf. one-point crossover.

Multimembered Evolution Strategy: All variants of evolution strategies that use a parent population size of $\mu > 1$ and therefore facilitate the utilization of recombination are summarized under the term multimembered evolution strategy.

Mutation: A change of the genetic material, either occurring in the germ path or in the gametes (generative) or in body cells (somatic). Only generative mutations affect the offspring. A typical classification of mutations distinguishes gene mutations (a particular gene is changed), chromosome mutations (the gene order is changed by translocation or inversion, or the chromosome number is changed by deficiencies, deletions or duplications), and genome mutations (the number of chromosomes or genomes is changed).

In evolutionary algorithms, mutations are either modeled on the phenotypic level (e.g., by using normally distributed variations with expectation zero for continuous traits) or on the genotypic level (e.g., by using bit inversions with small probability as an equivalent for nucleotid base changes).

Mutation Rate: The probability for the occurrence of a mutation during DNA-replication.

Natural Selection: The result of competitive exclusion as organisms fill the available finite resource space.

Object Variables: The parameters that are directly involved in the calculation of the objective function value of an individual.

Order: The order of a schema in genetic algorithm theory is given by the number of specified positions within the schema. The larger the order of a schema, the higher becomes its disruption probability by mutation.

Order Statistics: Given λ independent random variables with a common probability density function, their arrangement in nondecreasing order is called the order statistics of these random variables. The theory of order statistics provides many useful results regarding the moments (and other properties) of the members of the order statistics. In the theory of evolutionary algorithms, the order statistics are widely utilized to describe deterministic selection schemes such as the comma strategy and tournament selection.

Panmictic Population: A mixed population, in which any individual may be mated with any other individual with a probability that depends only on fitness. Most conventional evolutionary algorithms have panmictic populations.

Parse Tree: The syntactic structure of any program in computer programming languages can be represented by a so-called parse tree, where the internal nodes of the tree correspond with operators and leaves of the tree correspond with constants. Parse trees (or, equivalently, S-expressions) are the fundamental data structure in genetic programming, where recombination is usually implemented as a subtree-exchange between two different parse trees.

Phenotype: The behavioral expression of the genotype in a specific environment.

Plus Strategy: The notation $(\mu+\lambda)$-strategy describes a selection method introduced in evolution strategies and indicates that a parent population of μ individuals generates $\lambda \geq \mu$ offspring and all $\mu + \lambda$ individuals compete directly, such that the μ best out of parents and offspring are deterministically selected as parents of the next generation.

Population: A group of individuals that may interact with each other, e.g. by mating and offspring production. The typical population sizes in evolutionary algorithms range from one (for $(1 + 1)$-evolution strategies) to several thousands (for genetic programming).

Principle of Minimal Alphabets: The classical schema theory of genetic algorithms emphasizes the working mechanism of such algorithms as processing schemata. Given a fixed number of points in the search space, the principle of minimal alphabetes states that the number of schemata is maximized if an alphabet of cardinality two, i.e., a minimal alphabet, is used to represent search points.

Proportional Selection: Proportional selection is the basic selection mechanism used in genetic algorithms, which assigns selection probabilities in proportion to the relative fitness of an individual.

Punctuated Crossover: A crossover operator tested in canonical genetic algorithms to explore the potential for self-adaptation of the number of crossover points and their positions. To achieve this, the vector of object variables is extended by a crossover mask, where a one bit indicates the position of a crossover point in the object variable part of the individual. The crossover mask itself is subject to recombination and mutation to allow for a self-adaptation of the crossover operator.

Ranking Selection: In ranking selection methods, the selection probability of an individual does not depend on its absolute fitness as in case of proportional selection, but only on its relative fitness in comparison with the other population members, i.e., its rank when all individuals are ordered in increasing (or decreasing) order of fitness values.

Recombination: See crossover.

Scaling Function: Genetic algorithms typically require a scaling function to allow for the application of proportional selection, since this selection method assumes strictly positive fitness values and a maximization problem. Scaling functions typically employ a linear, logarithmic, or exponential mapping to achieve this.

Schema: A schema describes a subset of all binary vectors of fixed length that have similarities at certain positions. A schema is typically specified by a vector over the alphabet $\{0, 1, \#\}$, where the # denotes a "wildcard" matching both zero or one.

Schema Theorem: Theorem devised by Holland to give a lower bound for the expected number of instances of a schema that are represented in the next generation of a canonical genetic algorithm using one-point crossover, mutation, and proportional selection. Essentially, the schema theorem states that short (measured in terms of the schema's defining length), low-order, better-than-average schemata (also called building blocks) receive an exponentially increasing number of trials in the following generations.

Selection: Selection is the operator of evolutionary algorithms, modeled after the principle of natural selection, which is used to direct the search process towards better regions of the search space by giving preference to individuals of higher fitness for mating and reproduction. The most widely used selection methods include the

comma and plus strategy, ranking selection, proportional selection, and tournament selection.

Self-adaptation: The principle of self-adaptation facilitates evolutionary algorithms to learn their own strategy parameters on-line during the search, without any deterministic exogeneous control, by means of evolutionary processes in the same way as the object variables are modified. More precisely, the strategy parameters (such as mutation rates, variances, or covariances of normally distributed variations) are part of the individual and undergo mutation (recombination) and selection as the object variables do.

The biological analogy consists in the fact that some portions of the DNA code for mutator genes or repair enzymes, i.e., some partial control over the DNA's mutation rate is encoded in the DNA.

Species: A population of similarly constructed organisms, capable of producing fertile offspring. Members of one species occupy the same ecological niche.

Steady-State Selection: A selection scheme which does not use a generation-wise replacement of the population, but rather replaces one individual per iteration of the main recombine-mutate-select loop of the algorithm. Usually, the worst population member is replaced by the result of recombination and mutation, if the resulting individual represents a fitness-improvement compared to the worst population member. The mechanism corresponds with a $(\mu + 1)$-selection method in evolution strategies (cf. plus strategy).

Strategy Parameter: The control parameters of an evolutionary algorithm are often referred to as strategy parameters. The particular setting of strategy parameters is often critical to gain good performance of an evolutionary algorithm, and the usual technique of empirically searching for an appropriate set of parameters is not generally satisfying. Alternatively, some researchers try techniques of meta-evolution to optimize the strategy parameters, while in evolution strategies and evolutionary programming the technique of self-adaptation is successfully used to evolve strategy parameters in the same sense as object variables are evolved.

Tournament Selection: Tournament selection methods share the principle of holding tournaments between a number of individuals and selecting the best member of a tournament group for survival to the next generation. The tournament members are typically chosen uniformly at random, and the tournament sizes (number of individuals involved per tournament) are typically small, ranging from two to ten individuals. The tournament process is repeated μ times in order to select a population of μ members.

Two-Membered Evolution Strategy: The two membered or $(1 + 1)$-evolution strategy is an evolutionary algorithm working with just one ancestor individual. A descendant is created by means of mutation, and selection selects the better of ancestor and descendant to survive to the next generation. Cf. plus strategy.

Uniform Crossover: A crossover operator in genetic algorithms, which was originally defined to work on binary strings. The uniform crossover operator exchanges each bit with a certain probability between the two parent individuals. The exchange probability typically has a value of one half, but other settings are possible. Cf. discrete recombination.

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

$f : M \rightarrow \mathbb{R}$ 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., $\vec{x} \in \{0,1\}^\ell$.
- 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

$$f : \{0,1\}^\ell \to I\!R \quad . \tag{1}$$

Sticking to the binary representation, genetic algorithms often enforce the utilization of encoding and decoding functions $h : M \to \{0,1\}^\ell$ and $h' : \{0,1\}^\ell \to M$ that facilitate mapping solutions $\vec{x} \in M$ to binary strings $h(\vec{x}) \in \{0,1\}^\ell$ 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 $\vec{x} \in I\!R^n$ by a binary string $\vec{y} \in \{0,1\}^\ell$ as follows: the binary string is logically divided into n segments of equal length ℓ' (i.e., $\ell = n \cdot \ell'$), each segment $(x_1 \ldots x_{\ell'})$ is decoded to yield the corresponding integer value $\sum_{i=1}^{\ell'} x_i 2^{i-1}$, and the integer value is in turn linearly mapped to the interval $[u_i, v_i] \subset I\!R$ (corresponding to the ith segment of the binary string) of real values. Figure 1 illustrates this decoding process for string segments of length $\ell' = 9$ (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.

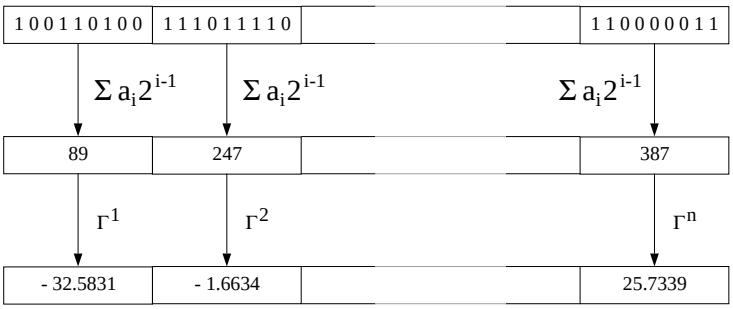

Figure 1: Decoding process used in canonical genetic algorithms for continuous search spaces. Γ^i denotes the linear mapping of an integer value $k \in \{0, \ldots, 2^{\ell'} - 1\}$ to the interval $[u_i, v_i]$, i.e., $\Gamma^i = u_i + \frac{v_i - u_i}{2^{\ell'} - 1} \cdot k$.

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 $p_m = 1/\ell$. 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.

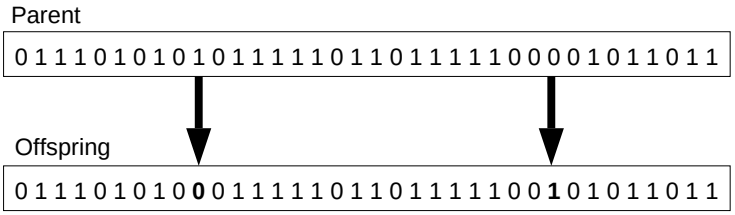

Figure 2: Mutation operator used in canonical genetic algorithms, with a bit inversion occurring at two random positions within the parent individual.

The mutation operator is illustrated in figure 2. In this example, two bits of the parent individual are inverted by mutation. In general, for $p_m = 1/\ell$, 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. 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 $p_c \in [0.75, 0.95]$.

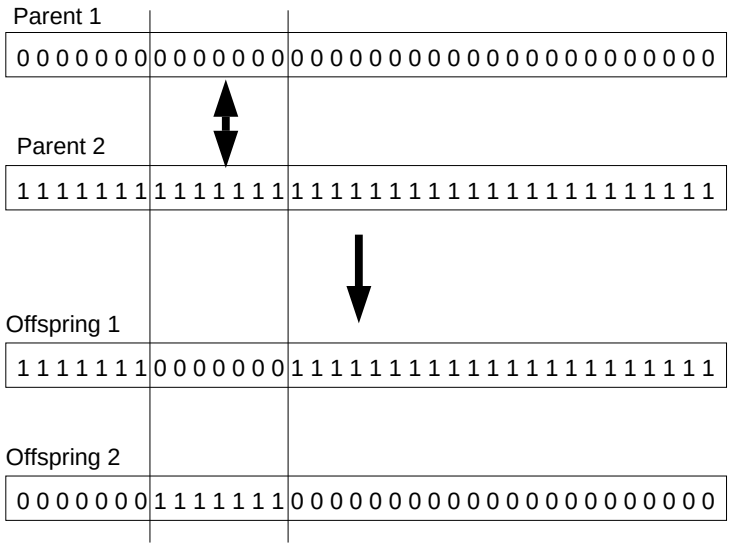

Figure 3: Two-point recombination operator used in canonical genetic algorithms. The two recombination positions are chosen randomly.

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

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

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 $\vec{x} \in I\!R^n$ plus some additional information, the so-called strategy parameters.
- No mating selection is used, i.e., $P'(t) = P(t)$.
- Assuming $\lambda \gg \mu$, the so-called (μ,λ)-selection operator (with $Q = \emptyset$) deterministically chooses the μ best solutions from $P''(t)$ to become $P(t+1)$. Alternatively, the (μ+λ)-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

$$f : M \subseteq I\!R^n \rightarrow I\!R \quad , \quad f(\vec{x}) \rightarrow \min \tag{3}$$

an individual of the evolution strategy contains the candidate solution $\vec{x} \in I\!R^n$ 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 $\vec{x} \in I\!R^n$ (the solution), $\vec{\sigma} \in I\!R^{n_\sigma}$ (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 $n_\sigma \in \{1, \ldots, n\}$ 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.

3.2 Mutation

The mutation in evolution strategies works by adding a normally distributed random vector $\vec{z} \sim N(\vec{0}, \mathbf{C})$ with expectation vector $\vec{0}$ and covariance matrix $\mathbf{C}^{-1}$, where the covariance matrix is described by the mutated strategy parameters of the individual. Depending on the amount of strategy parameters incorporated into the representation of an individual, the following main variants of mutation and self-adaptation can be distinguished:

- $n_\sigma = 1$, $n_\alpha = 0$: The standard deviation (σ) for all object variables is identical and all object variables are mutated by adding normally distributed random numbers with

$$\sigma' = \sigma \cdot \exp(\tau_0 \cdot N(0,1)) \tag{4}$$
$$x_i' = x_i + \sigma' \cdot N_i(0,1) \quad , \tag{5}$$

  where $\tau_0 \propto (\sqrt{n})^{-1}$. Here, $N(0,1)$ denotes a value sampled from a normally distributed random variable with expectation zero and variance one. The notation $N_i(0,1)$ indicates the random variable to be sampled anew for each setting of the index i.

- $n_\sigma = n$, $n_\alpha = 0$: All object variables have their own, individual standard deviation σ_i, which determines the corresponding modification according to

$$\sigma_i' = \sigma_i \cdot \exp(\tau' \cdot N(0,1) + \tau \cdot N_i(0,1)) \tag{6}$$
$$x_i' = x_i + \sigma_i' \cdot N(0,1) \quad , \tag{7}$$

  where $\tau' \propto (\sqrt{2n})^{-1}$ and $\tau \propto (\sqrt{2\sqrt{n}})^{-1}$.

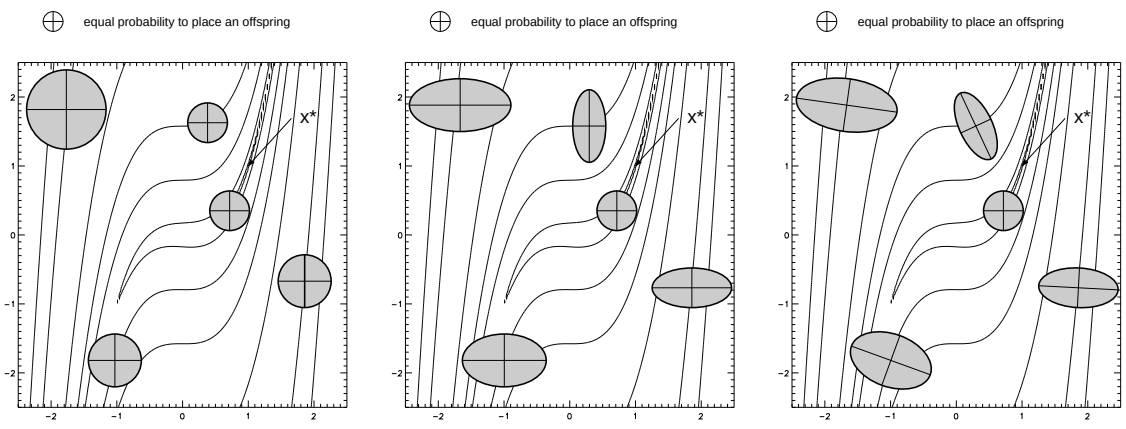

Figure 4: Schematic visualization of the three different types of self-adaptive mutation in evolution strategies. Left: $n_\sigma = 1$, middle: $n_\sigma = 2$, right: $n_\sigma = 2$, $n_\alpha = 1$.

- $n_\sigma = n$, $n_\alpha = n \cdot (n-1)/2$: The vectors $\vec{\sigma}$ and $\vec{\alpha}$ represent the complete covariance matrix of the n-dimensional normal distribution, where the covariances are given by rotation angles α_j describing the coordinate rotations necessary to transform an uncorrelated mutation vector into a correlated one. The details of this mechanism can be found in the literature. The mutation is performed according to

$$\sigma_i' = \sigma_i \cdot \exp(\tau' \cdot N(0,1) + \tau \cdot N_i(0,1)) \quad (8)$$

$$\alpha_j' = \alpha_j + \beta \cdot N_j(0,1) \quad (9)$$

$$\vec{x}' = \vec{x} + N(\vec{0}, \mathbf{C}(\vec{\sigma}', \vec{\alpha}')) \quad (10)$$

  where $N(\vec{0}, \mathbf{C}(\vec{\sigma}', \vec{\alpha}'))$ denotes the correlated mutation vector and $\beta \approx 0.0873$.

It should be noted that strategy parameters are mutated first, and the mutated strategy parameters are used to mutate the object variables. Selection implicitly rewards or penalizes strategy parameters because of their impact on fitness values only, i.e., advantageous strategy parameters are likely to survive because of their improving effect on fitness.
The amount of information included into the individuals by means of the self-adaptation principle increases from the simple case of one standard deviation up to the order of n^2 additional parameters in case of *correlated mutations*, which reflects an enormous degree of freedom for the *internal models* – represented by strategy parameter sets – of the individuals. This growing degree of freedom often enhances the global search capabilities of the algorithm at the cost of the expense in computation time, and it also reflects a shift from the precise *adaptation* of a few strategy parameters (as in case of $n_\sigma = 1$) to the exploitation of a large *diversity* of strategy parameters.
Figure 4 illustrates the growing degree of freedom as the number of strategy parameters is increased. Each of the three figures shows a two-dimensional (i.e., $n = 2$) hypothetical objective function topology, including isolines of equal objective function value and the location of a global optimum $\vec{x}^*$. The gray-shaded circles and ellipsoids correspond to individuals (located at the centerpoints of the circles and ellipsoids) and their corresponding

probability distribution to place an offspring. For $n_\sigma = 1$, as shown in the left part of the figure, all distributions are spherically symmetric, and only the radius of the circles is individually different. For $n_\sigma = 2$, as in the middle figure, step sizes along one dimension might be different from the other dimension, such that preference directions of search can be adjusted. In case of the correlated mutation with $n_\sigma = 2$, $n_\alpha = 1$, as shown in the right figure, the ellipsoids can rotate and therefore allow an adjustment of arbitrary preference directions regardless of the coordinate system. One of the main design parameters to be fixed for the practical application of the evolution strategy concerns the choice of n_σ and n_α, i.e., the amount of self-adaptable strategy parameters required for the problem. It is clear that increasing the amount of strategy parameters increases the degree of freedom for the search process as well, at the cost of rapidly growing computational effort and decreasing convergence speed. In general, it is advisable to start with simple strategies first and gradually increase the number of strategy parameters if required.

3.3 Recombination

In evolution strategies recombination is incorporated into the main loop of the algorithm as the first variation operator and generates a new intermediate population of λ individuals by λ-fold application to the parent population, creating one individual per application from ϱ ($1 \leq \varrho \leq \mu$) individuals. Normally, $\varrho = 2$ or $\varrho = \mu$ (so-called global recombination) are chosen. The recombination types for object variables and strategy parameters in evolution strategies often differ from each other, and typical examples are *discrete recombination* (random choices of single variables from parents, comparable to uniform crossover in genetic algorithms) and *intermediary recombination* (arithmetic averaging). A typical setting of the recombination consists in using discrete recombination for object variables and global intermediary recombination for strategy parameters. An example of discrete recombination on object variables and intermediate recombination on strategy parameters is shown in figure 5.

3.4 Selection

Essentially, the evolution strategy offers two different variants for selecting candidate solutions for the next iteration of the main loop of the algorithm: (μ,λ)-selection and (μ+λ)-selection.
The notation (μ, λ) indicates that μ parents create $\lambda > \mu$ offspring by means of recombination and mutation, and the best μ offspring individuals are deterministically selected to replace the parents (in this case, $Q = \emptyset$ in algorithm 1). Notice that this mechanism allows that the best member of the population at generation $t + 1$ might perform *worse* than the best individual at generation t, i.e., the method is not *elitist*, thus allowing the strategy to accept temporary deteriorations that might help to leave the region of attraction of a local optimum and reach a better optimum. Moreover, in combination with the self-adaptation of strategy parameters, (μ,λ)-selection has demonstrated clear advantages over its competitor, the (μ+λ) method. For this reason, the (μ,λ)-evolution strategy is typically preferred.

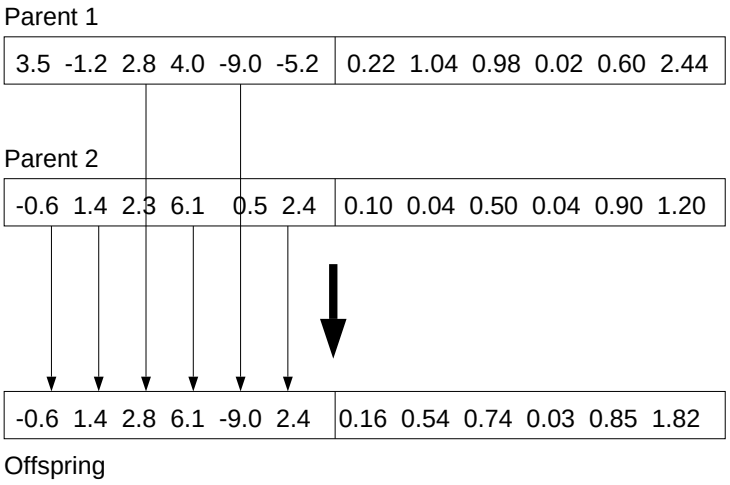

Figure 5: Illustration of the recombination operator for evolution strategies, with $n = n_\sigma = 6$. Here, discrete recombination on object variables and intermediate recombination on strategy parameters are shown. In many implementations, the offspring is not necessarily created from object variable and strategy parameter sets that originate from the same two parents.

In contrast, the $(\mu+\lambda)$-strategy selects the μ survivors from the union of parents and offspring, such that a monotonic course of evolution is guaranteed ($Q = P(t)$ in algorithm 1). This method is typically used in a steady-state setting (self-adaptation does not work properly, if $\gamma < 1$), or under circumstances where fitness deteriorations from one generation to the next are strictly unacceptable.

4 Evolutionary Programming

Again using the general evolutionary algorithm outline given in algorithm 1, the main properties of evolutionary programming as refined in the early 90es can be summarized as follows:

- Individuals are represented as real-valued vectors, consisting of object variable vectors $\vec{x} \in I\!R^n$ plus some additional information, the so-called strategy parameters.
- No mating selection is used, i.e., $P'(t) = P(t)$.
- Parent and offspring population sizes are identical, i.e., $\lambda = \mu$. Selection is implemented as a randomized process taking parents and offspring into account (i.e., $Q = P(t)$).
- The mutation operator is implemented by means of normally distributed variations based on adaptable step sizes (i.e., typically variances of the normal distribution).

- Recombination is not used.

In the following, we will again summarize the main aspects of this instance of evolutionary algorithms.

4.1 The Structure of Individuals

Similar to evolution strategies, also modern variants of evolutionary programming use the concept of self-adaptive strategy parameters in combination with real-valued candidate solutions $\vec{x} \in I\!R^n$. Again, the strategy parameters represent a vector $\vec{\sigma} = (\sigma_1, \ldots, \sigma_n)$ of standard deviations of a normal distribution used for mutating the object variables. The self-adaptation idea was introduced independently from evolution strategies (but 20 years later) into evolutionary programming.

4.2 Mutation

The choice of a logarithmic normal distribution as defined in equation (6) for the modification of the standard deviations σ_i is presently also acknowledged in evolutionary programming literature and this method has substituted the original additive self-adaptation mechanism where

$$\sigma_i' = \sigma_i \cdot (1 + \alpha \cdot N(0,1)) \tag{11}$$

(with a setting of $\alpha \approx 0.2$).

4.3 Recombination

Concerning evolutionary programming, a rash statement based on the common understanding of the contending structures as individuals would be to claim that evolutionary programming simply does not use recombination. Rather than focusing on the mechanism of sexual recombination, however, Fogel argues that one may examine and simulate its functional effect and correspondingly interpret a string of symbols as a reproducing population or species, thus making recombination a non-issue. While these are more philosophical reasons for not incorporating recombination into the algorithm, it is sometimes also argued that this distinguishing feature is chosen purposefully to demonstrate the clear differences between genetic algorithms – which focus strongly on recombination – and evolutionary programming, which drops it completely.
At present, the role of recombination in evolutionary algorithms is not perfectly understood. The advantages or disadvantages of recombination for a particular objective function can hardly be assessed in advance, and certainly no generally useful setting of recombination operators (such as the discrete recombination of object variables and global intermediary of strategy parameters as sometimes claimed for evolution strategies) exists. Recently, Kursawe has impressively demonstrated that, using an inappropriate setting of the recombination operator, the (15,100)-evolution strategy with n self-adaptable variances might even diverge on a sphere model for $n = 100$. His work shows that the appropriate choice of the recombination operator not only depends on the objective function topology, but also on the dimension of the objective function and the number of

strategy parameters incorporated into the individuals. Only recently, Rechenberg and Beyer presented first results concerning the convergence velocity analysis of global recombination in case of the sphere model. These results clarify that, for using one (rather than n as in Kursawe's experiment) optimally chosen standard deviation σ, a μ-fold speedup is achieved by both recombination variants.

4.4 Selection

A minor difference between evolutionary programming and evolution strategies consists in the choice of a probabilistic variant of $(\mu + \lambda)$-selection in evolutionary programming, where each solution out of offspring and parent individuals is evaluated against $q > 1$ (typically, $q \leq 10$) other randomly chosen solutions from the union of parent and offspring individuals ($Q = P(t)$ in algorithm 1). For each comparison, a "win" is assigned if an individual's score is better or equal to that of its opponent, and the μ individuals with the greatest number of wins are retained to be parents of the next generation. As shown by Bäck, this selection method is a probabilistic version of $(\mu + \lambda)$-selection which becomes more and more deterministic as the number q of competitors (the tournament size) is increased. Whether or not a probabilistic selection scheme should be preferable over a deterministic scheme remains an open question.

5 Genetic Programming

Genetic programming applies evolutionary search to the space of tree structures which may be interpreted as computer programs in a language suitable to modification by mutation and recombination. The dominant approach to genetic programming uses (a subset of) LISP programs (S-expressions) as genotype space, but other programming languages including machine code are also used.

5.1 The Structure of Individuals

Figure 6 gives an example of an arithmetic expression in tree structure, which reads $x \cdot (2.5 + y)$ in mathematical notation and `(* x (+ 2.5 y))` in LISP notation. As a standard convention, tree structures are evaluated by repeatedly evaluating the leftmost node for which all inputs are available.
Characteristic to individuals represented by tree structures is the distinction between internal nodes (labeled with unary, binary, or higher arity operators, also called functions) and terminal nodes, which are labeled with constants, variable names, or other entities that can be directly evaluated or executed. For initialization purposes, the terminal and function sets as well as the maximum allowed tree depth (the depth of a node is the minimal number of nodes on a path from the root node to the actual node) must be specified. The initialization procedure then grows trees by randomly selecting nodes from the function set and the terminal set, up to the maximum depth specified.

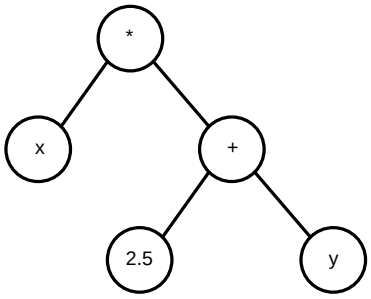

Figure 6: A tree structure (maximum depth 3, terminals $T = \{x, y, 2.5\}$, functions $F = \{*, +\}$) in standard genetic programming.

5.2 Mutation

At present, various mutation operators have been proposed for tree-based genetic programming and are used with low probability to maintain diversity in the population. All mutation operators have in common the fact that one parent individual is mutated to create an offspring individual. The mutation can consist in a single node echange against a random node of the same class (point mutation), a terminal exchange against a randomly generated subtree (expansion mutation), a subtree exchange against a random terminal (collapse subtree mutation), a subtree exchange against a new random subtree, a permutation of node arguments, and others. The simplest example, a point mutation, is illustrated in figure 7 for the simple tree of figure 6.

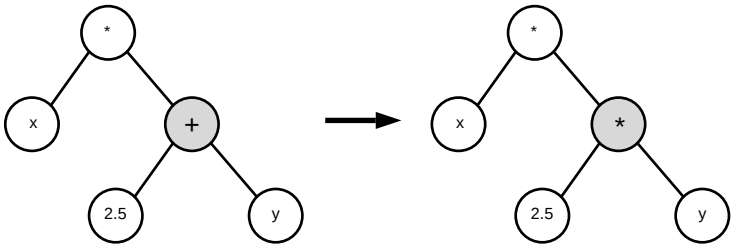

Figure 7: Effect of a point mutation in standard genetic programming.

5.3 Recombination

The crossover operator combines the genetic material of two parent trees by swapping a part of one parent with a part of the other. An example of the crossover operator is shown in figure 8.

For the crossover operator, two parent trees are chosen from the population, based on the mating selection operator `select1` (e.g., fitness-proportional selection). Then, in each parent a random subtree is selected, highlighted in figure 8 by grey shaded nodes.

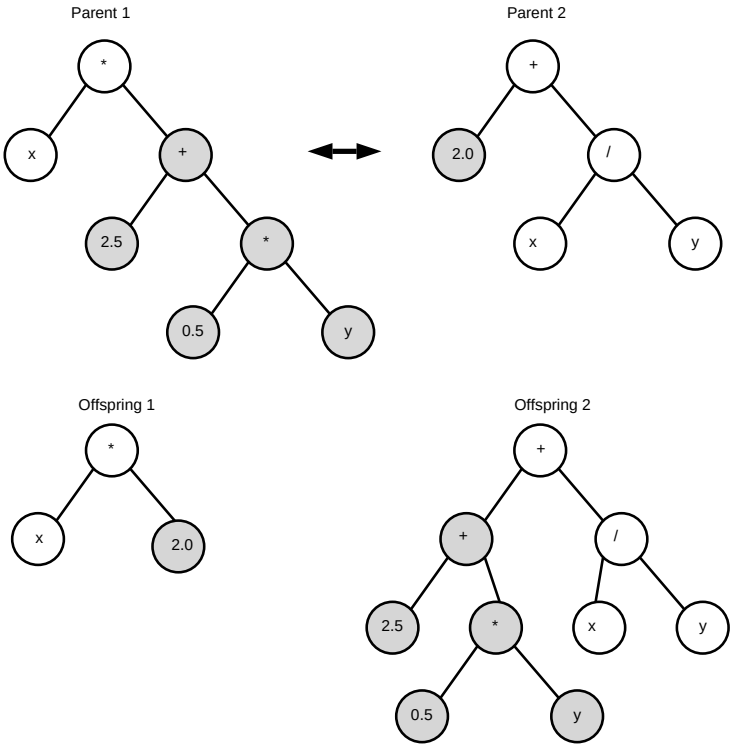

Figure 8: Effect of tree-based crossover in genetic programming.

Finally, the selected subtrees are swapped between the two parents, to yield the offspring individuals as shown at the bottom of figure 8.

5.4 Selection

As the selection operator in evolutionary algorithms is completely independent of the particular representation and variation operators used, any selection operator can in principle also be used for genetic programming. Although for quite some while, reflecting the historical roots of genetic programming, proportional selection was preferred in genetic programming, tournament selection has now become the mainstream method also in genetic programming. It combines the advantages of efficient implementation and reasonably strong selective pressure, such that evolution is considerably accelerated.

6 Theory of Evolutionary Algorithms

Unlike other evolutionary algorithms, e.g. genetic algorithms, the theory of evolution strategies has from the very beginning focused on the questions of *convergence velocity* and *convergence reliability* of these algorithms. Loosely speaking, the former concentrates on the speed of the algorithm when a local optimum is approached (and therefore it

provides insight into the local behaviour of an evolutionary algorithm) while the latter concentrates on proving that the algorithm is capable of finding the global optimum of the objective function. The convergence velocity analysis requires strong simplifications regarding the objective functions that can be analyzed – so far, the convex case can be handled – while the convergence reliability (i.e., global convergence with probability one) analysis yields a result for $t \to \infty$ (i.e., an asymptotical result as the running time of the algorithm goes to infinity) independently of the objective function, i.e., a result which is useless for all practical purposes.
These two types of analysis provide important benchmark information regarding the properties of evolutionary algorithms as an optimization algorithm: With respect to the convergence velocity, they should be competitive with gradient-based methods and with respect to convergence reliability, they should prove to be global optimization algorithms, i.e., have the property of global convergence with probability one. The following sections deal with these aspects of evolutionary algorithms and discuss also some related topics such as selection schemes and the transfer of theoretical results to genetic algorithms.

6.1 Convergence Velocity

Traditionally, evolution strategies have been analyzed in terms of their *convergence velocity*, i.e., the expected improvement in fitness (12) or distance from the optimum $\vec{x}^*$ (13) between two consecutive generations:

$$\varphi = \mathbf{E}(|f(\vec{x}^*) - f(\vec{x}_t)| - |f(\vec{x}^*) - f(\vec{x}_{t+1})|) \tag{12}$$

$$\varphi = \mathbf{E}(\|\vec{x}^* - \vec{x}_t\| - \|\vec{x}^* - \vec{x}_{t+1}\|) \tag{13}$$

The second alternative, distance to the optimum, has been used in evolution strategies with enormous success to analyze algorithmic instances such as the (1+1)-, (1+λ)-, and (1,λ)-, (μ,λ)-strategies (with $\mu > 1$, but without recombination for the sake of theoretical analysis) and (μ/μ,λ)-strategies (with global intermediary or global discrete recombination) on simple quadratic objective functions, namely the sphere model $f(\vec{x}) = f((x_1, \ldots, x_n)^T) = \sum_{i=1}^{n}(x_i - x_i^*)^2 = R^2$ for the assumption of one fixed mutation step size σ and sufficiently large dimensionality n. Without going into the details here, it is worthwhile to mention that for comma-strategies the results can usually be expressed as simple quadratic equations such as

$$\varphi'_{(1,\lambda)} = c_{1,\lambda} \cdot \sigma' - \frac{\sigma'^2}{2} \tag{14}$$

for a (1,λ)-strategy, where $\varphi' = \varphi n/R$ and $\sigma' = \sigma n/R$ denote normalized variables, R is the current distance of the parent $\vec{x}$ to the optimum location $\vec{x}^*$, and $c_{1,\lambda}$ denotes the *progress coefficient* of the strategy. For a (1,λ)-strategy, this constant depends only on λ and characterizes the selective pressure of the (1,λ)-selection method. It is defined as the expectation of the largest of λ independent random variables Z_i with identical, standardized and normalized Gaussian distribution $Z_i \sim N(0,1)$, which are rearranged in increasing order and relabeled by $Z_{1:\lambda} \leq Z_{2:\lambda} \leq \ldots \leq Z_{\lambda:\lambda}$:

$$c_{1,\lambda} = \mathbf{E}(Z_{\lambda:\lambda}) = \int_{-\infty}^{\infty} z \cdot \frac{d}{dz}[\Phi(z)]^\lambda dz \; . \tag{15}$$

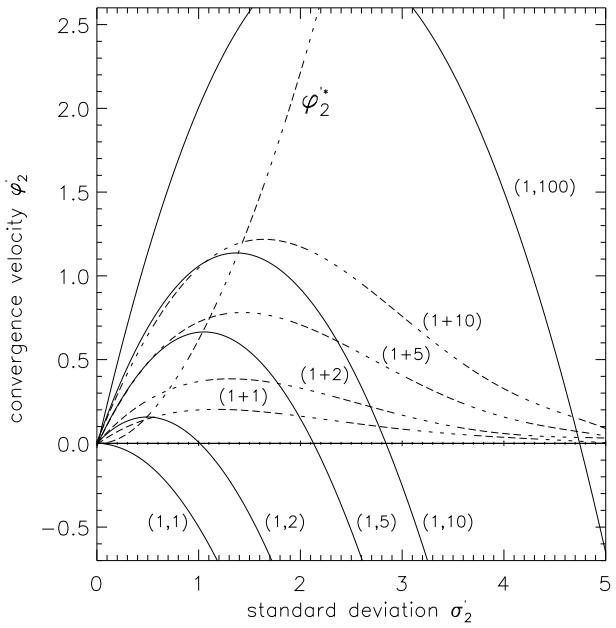

Figure 9: Convergence velocity as a function of the step size for various evolution strategies.

Here, Φ denotes the distribution function of the Gaussian distribution. The values of $c_{1,\lambda}$ are extensively studied and tabulated in the theory of order statistics.
Notice that equation (14) reflects a *progress gain* $\sigma' c_{1,\lambda}$ as well as a *progress loss* $\sigma'^2/2$, and these two terms have to be kept in balance (either by increasing the gain or by decreasing the loss) in order to achieve positive convergence velocity.
From equation (14), the optimal standard deviation

$$\sigma^* = c_{1,\lambda} \cdot \frac{R}{n} \tag{16}$$

and the resulting maximal relative progress

$$\varphi^*_{(1,\lambda)} = \frac{c^2_{1,\lambda}}{2} \cdot \frac{R}{n} \tag{17}$$

are easily obtained. The *convergence velocity* φ (or its normalized counterpart $\varphi' = \varphi \cdot n/R$) is related to the relative progress according to $\varphi = \varphi' R/n$, such that, using the asymptotic result $c_{1,\lambda} \approx \sqrt{2 \ln \lambda}$, one finally obtains

$$\varphi'^*_{(1,\lambda)} \approx \ln \lambda \quad . \tag{18}$$

For the details of this derivation, the reader is referred to the literature.
Figure 9 illustrates these results for (1,λ)-strategies and (1+λ)-strategies (for which no closed analytical expressions can be derived) by plotting convergence velocity as a function

of mutation step size for different values of λ. Also the difference between the selection schemes becomes clear in this figure, as the (1+λ)-strategy guarantees positive convergence velocity, which however quickly approaches zero as σ increases beyond a certain limit. Keeping σ within this window of positive convergence velocity ($\sigma' < 2 \cdot c_{1,\lambda}$ for the (1,λ)-strategy) is the main task of any workable and competitive parameter control scheme. In evolution strategies, this problem is very successfully solved by the principle of self-adaptation, which uses evolutionary mechanisms to adapt these parameters individually. For an in-depth overview of this method, refer to the literature.
In case of the (μ,λ)-ES without recombination, the progress coefficients $c_{\mu,\lambda}$ are generalized to reflect the expectation of the average of the μ best offspring, i.e.,

$$c_{\mu,\lambda} = \frac{1}{\mu} \cdot \sum_{i=\lambda-\mu+1}^{\lambda} \mathbf{E}(Z_{i:\lambda}) \, . \tag{19}$$

For these strategies, the convergence velocity is given by the equation

$$\varphi'_{(\mu,\lambda)} = \sigma' c_{\mu,\lambda} - \frac{1}{2}\sigma'^2 \quad , \tag{20}$$

and one obtains the results

$$\sigma'^* = c_{\mu,\lambda} \tag{21}$$

and

$$\varphi'^*_{(\mu,\lambda)} = \frac{1}{2} c^2_{\mu,\lambda} \quad , \tag{22}$$

which asymptotically yields

$$\varphi'^*_{(\mu,\lambda)} \sim \ln \frac{\lambda}{\mu} \tag{23}$$

($c_{\mu,\lambda} \approx \mathcal{O}(\sqrt{\ln \lambda/\mu})$). This result reflects the dominating impact of the *selective pressure* λ/μ for the convergence velocity.
Taking also global recombination (i.e., with $\varrho = \mu$ parents involved in the creation of a single offspring) into account, Beyer and Rechenberg recently derived the equation

$$\varphi'_{(\mu,\lambda),\varrho=\mu_I} = \sigma' c_{\mu/\mu,\lambda} - \frac{1}{\mu}\frac{\sigma'^2}{2} \quad , \tag{24}$$

resulting in the optimal step size

$$\sigma'^*_I = \mu c_{\mu/\mu,\lambda} \tag{25}$$

and the corresponding convergence velocity

$$\varphi'^*_{(\mu,\lambda),\varrho=\mu_I} = \frac{\mu}{2} \cdot c^2_{\mu/\mu,\lambda} \tag{26}$$

for intermediary recombination on the sphere model.
With the asymptotical result $c_{\mu/\mu,\lambda} \approx \sqrt{\ln \lambda/\mu}$ (the relation $0 \leq c_{\mu/\mu,\lambda} \leq c_{\mu,\lambda} \leq c_{1,\lambda} < \sqrt{2 \ln \lambda}$ holds for the various progress coefficients), a μ-fold speedup of intermediary recombination as opposed to no recombination is obtained:

$$\varphi'^*_{(\mu,\lambda),\varrho=\mu_I} \approx \mu \ln \frac{\lambda}{\mu} \quad . \tag{27}$$

Surprisingly, for discrete recombination Beyer obtained the same result for the optimal convergence velocity φ'^* from the progress equation

$$\varphi'^*_{(\mu,\lambda),\varrho=\mu_D} = \sqrt{\mu}\sigma' c_{\mu/\mu,\lambda} - \frac{1}{2}\sigma'^2 \quad , \tag{28}$$

but the optimal step size

$$\sigma'^*_D = \sqrt{\mu} c_{\mu/\mu,\lambda} \tag{29}$$

turns out to be a factor of $\sqrt{\mu}$ smaller.
Resulting from his mathematical analysis, Beyer is able to explain the beneficial effect of recombination by the fact that recombination reduces the progress loss term in the quadratic equation for φ'. In other words, recombination serves as a statistical error correction method, reducing the impact of the harmful part of mutations. This effect, called *genetic repair* by Beyer, clarifies that the conjecture of the building block hypothesis often emphasized in the theory of genetic algorithms does not hold for evolution strategies. The building block hypothesis essentially claims that the combination of good partial solutions explains the advantage of recombination. Also for discrete recombination, Beyer concludes in contrary to the classical building block hypothesis (see section 6.3) that it works by implicitly performing a genetic repair, such that (by analogy with uniform crossover in genetic algorithms), it is very likely that his results will also fundamentally change the direction of further theoretical research regarding genetic algorithms. Figure 10 illustrates the effects of discrete and intermediary recombination by showing all possible offspring individuals in case of a 2-dimensional convex function and three parents (black circles). It is obvious from this simple example that, with reasonably high probability, both recombination operators create an improved offspring individual.

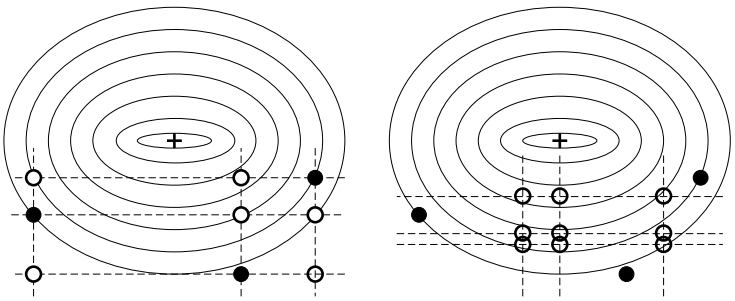

Figure 10: Possible recombinants (circles) created by discrete recombination (left) and intermediary recombination (right) for $n = 2$ from three parents, shown as black circles.

To conclude, we would like to emphasize that the local convergence velocity analysis of evolution strategies is a constructive analysis facilitating statements about the optimal working regime of the algorithm. In particular, it allows to evaluate the performance of parameter control methods such as self-adaptation with respect to the optimal schedule for σ as obtained from the theory.

6.2 Convergence Velocity Analysis of Genetic Algorithms

Starting in the early 90s, the constructive aspects of this theory were transferred to binary search spaces to facilitate the analysis of simple mutation-based variants of genetic algorithms such as the (1+1)-GA. By analogy with the (1+1)-evolution strategy, one parent individual generates one offspring by mutation only, and the better of the two individuals (the offspring in case of a tie) survives to become parent of the next iteration. The differences to an evolution strategy consists in the facts that individuals are binary vectors, mutation inverts each bit independently with probability $p_m \in (0,1)$, and the mutation rate remains constant rather than undergoing some adaptation process.
Here, we give a brief outline of the basic idea for the analysis, which focuses on the bit-counting function $f : \{0,1\}^n \to I\!R$, $f(\vec{x}) = \sum_{i=1}^{n} x_i$ for simplicity, and a standard bit inversion mutation operator with a mutation rate p_m per bit. Assuming a maximization task, the convergence velocity is then defined by means of the expected improvement in fitness value as follows:

$$\varphi_{(1+1)} = \sum_{k=0}^{n-f(\vec{x})} k \cdot p(k) \; , \tag{30}$$

where $p(k)$ denotes the probability that mutation improves the fitness value of the vector $\vec{x}$ by a value of k, i.e., $p(k) = P\{f(\texttt{mutate}(\vec{x})) = f(\vec{x}) + k\}$. Knowing these transition probabilities, the (1+1)-algorithm can then be modeled as a finite homogenous Markov chain with a special triangular transition matrix and $n+1$ different states corresponding with the different fitness values. The exact transition probabilities are given by the expression

$$p(k) = \sum_{i=0}^{f(\vec{x})} \binom{f(\vec{x})}{i} \binom{n - f(\vec{x})}{i+k} p^{2i+k} (1-p)^{n-2i-k} \tag{31}$$

and facilitate numerical investigations of the Markov chain with respect to properties such as the optimal mutation rate (maximizing convergence velocity) and the expected time to absorption (i.e., time to find the optimum).
The results clarify that the optimal mutation rate for the bit counting problem is inversely proportional to the dimensionality of the problem and depends on the actual fitness value $f(\vec{x})$, i.e.

$$p^* \approx \frac{1}{2(f(\vec{x}) + 1) - n} \; . \tag{32}$$

Moreover, an analytical treatment of the problem becomes possible by neglecting backward mutations, i.e., those mutations that change correct bits back to incorrect ones, resulting in an approximation of p^* as $p^* \approx 1/n$, which is a reasonable general-purpose heuristic for genetic algorithms.
While the generalization of this kind of convergence velocity analysis to (1,λ)- and (μ,λ)-genetic algorithms was formalized without obtaining a closed expression for the optimal mutation rate, Beyer recently presented an approximation for the (1,λ)-genetic algorithm applied to the bit counting function:

$$p^* \approx \frac{c_{1,\lambda}^2}{4 \cdot (2 \cdot f(\vec{x})/f^* - 1)^2} \cdot \frac{1}{n} \; , \tag{33}$$

where $f^* = n$ denotes the optimal fitness value, $f(\vec{x}) > n/2$, and $n \to \infty$ are assumed. Even though these results were derived for the simple bit counting problem, they can serve at least as lower bounds for the mutation rate in case of more complex objective functions and are therefore of interest for any application of genetic algorithms to binary optimization problems. Using the familiar asymptotic approximation

$$c_{1,\lambda} \approx \sqrt{2 \cdot \ln \lambda} \tag{34}$$

for $\lambda \to \infty$ allows for a further simplification of this expression for the (1,λ)-genetic algorithm, yielding

$$p^* \approx \frac{\ln \lambda}{2 \cdot (2 \cdot f(\vec{x})/f^* - 1)^2} \cdot \frac{1}{n} \ , \tag{35}$$

which can further be approximated in the final stage of the search, when $f(\vec{x})$ is relatively close to its optimum value f^*, as

$$p^* \approx \frac{1}{2n} \cdot \ln \lambda \ . \tag{36}$$

Obviously, these values could easily be used to parameterize the corresponding genetic algorithm instance, though for arbitrary objective functions they can only serve as an indication for a good setting rather than a perfect choice.
Based on the convergence velocity analysis outlined above, plots of the (numerically determined) convergence velocity of (1,λ)-genetic algorithms as a function of the mutation rate demonstrate that the qualitative behaviour is identical to that in case of evolution strategies: As p_m increases, φ also increases up to a maximum, and then quickly decreases and becomes negative. As λ is increased, both φ^* and p^* increase. We conjecture that the fundamental evolutionary behaviour of these algorithms is identical, even though different search spaces are involved – the unifying theory, however, has not yet been developed completely.

6.3 Classical Genetic Algorithm Theory

The essentials of classical genetic algorithm theory, the so-called schema theory, are derived by viewing a genetic algorithm 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*

$$f(H^t) = \frac{1}{m(H^t)} \sum_{a_i \in H^t} f(a_i) \tag{37}$$

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

$$m(H^{t+1}) = m(H^t) \cdot f(H^t)/\bar{f}^t \ . \tag{38}$$

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 genetic algorithms:

$$m(H^{t+1}) \quad \geq \quad m(H^t) \cdot \frac{f(H^t)}{\bar{f}^t} \cdot \left(1 - p_c \frac{\delta(H^t)}{l-1}\right) (1-p_m)^{\sigma(H^t)} \tag{39}$$

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 genetic algorithm and therefore supports the hyperplane sampling process. Building blocks and their combination to form longer and longer useful substrings have been claimed the most important working mechanism of a genetic algorithm. This kind of theoretical view on the working mechanism of genetic algorithms has dominated the field since the early work of John Holland. Since about 1995, however, it is widely accepted that more constructive theoretical approaches are necessary to understand genetic algorithms, such that Markov chains and dynamical systems theory as well as probability theory are now widely used.

6.4 Global Convergence Properties

In this section, only the most basic result regarding global convergence of evolution strategies is presented. This result holds for any type of elitist strategy, i.e., $(\mu+\lambda)$-strategy. Before summarizing the convergence results, some basic definitions and assumptions are to be made. An optimization problem of the form

$$f^* := f(x^*) := \min\{f(x) | \, x \in M \subseteq I\!R^n\} \tag{40}$$

is called a *regular global optimization problem* iff

(A1) $f^* > -\infty$,
(A2) $x^* \in \mathrm{int}(M) \neq \emptyset$ and
(A3) $\mu(L_{f^*+\epsilon}) > 0$,

where $\mu(.)$ denotes the Lebesgue measure and $\mathrm{int}(M)$ the interior of set M.
Here, f is called the *objective function*, M the *feasible region*, f^* the *global minimum*, x^* the *global minimizer* or *solution* and $L_a := \{x \in M | \, f(x) < a\}$ the *lower level set* of f. The

first assumption makes the problem meaningful whereas the second assumption is made to facilitate the analysis. The last assumption skips those problems where optimization is a hopeless task. The proof of global convergence for evolution strategy algorithms has been given by several authors independently. Although the conditions are slightly different among the papers each proof is based on the Borel–Cantelli Lemma.
Let $p_t := \mathcal{P}\{X_t \in L_{f^*+\epsilon}\}$ be the probability to hit the level set $L_{f^*+\epsilon}$, $\epsilon > 0$, at step t. If

$$\sum_{t=1}^{\infty} p_t = \infty \tag{41}$$

then $f(X_t) - f^* \to 0$ a.s. for $t \to \infty$ or equivalently

$$\mathcal{P}\{\lim_{t\to\infty}(f(X_t) - f^*) = 0\} = 1 \tag{42}$$

for any starting point $x_0 \in M$. □
Let C be the support of stationary probability distribution p_Z (i.e., the standard deviation of mutations is considered to be constant) of the random vectors generated by the mutation operator. Then the following holds: $M \subseteq C$ and M bounded $\Rightarrow$ L_a bounded $\forall\, a \leq f(x_0) \Rightarrow p_t \geq p_{min} > 0$ for all $t > 0 \Rightarrow \liminf_{t\to\infty} p_t > 0 \Rightarrow$ (41)
Of course, the above theorem is only of academic interest because we do not have unlimited time to wait. But if condition (41) is not fulfilled then one may conclude that the probability to obtain the global minimum for any starting point $x_0 \in M$ with increasing t is zero. A unifying and more general proof of global convergence for evolutionary algorithms in general search spaces has been given more recently.

7 Applications

Practical application problems in fields as diverse as engineering, natural sciences, economics, and business (to mention only some of the most prominent representatives) often exhibit a number of characteristics that prevent the straightforward application of standard instances of evolutionary algorithms. Typical problems encountered when developing an evolutionary algorithm for a practical application include the following:

1. A suitable representation and corresponding operators need to be developed when the canonical representation is different from binary strings or real-valued vectors.

2. Various constraints need to be taken into account by means of a suitable method (ranging from penalty functions to repair algorithms, constraint-preserving operators, and decoders).

3. Expert knowledge about the problem needs to be incorporated into the representation and the operators in order to guide the search process and increase its convergence velocity — without running into the trap, however, to get confused and misled by expert beliefs and habits which might not correspond with the best solutions.

4. An objective function needs to be developed, often in cooperation with experts from the particular application field.

5. Technical problems concerning software interfaces (between optimizer and existing simulation software), licenses and compatibility need to be solved.

6. The parameters of the evolutionary algorithm need to be set (or tuned), and the feasibility of the approach needs to be assessed by comparing the results to expert solutions (used so far) or, if applicable, solutions obtained by other algorithms.

Most of these topics require experience with evolutionary algorithms as well as cooperation between the application's expert and the evolutionary algorithm expert, and only few general results are available to guide the design of the algorithm (e.g., representation-independent recombination and mutation operators, the requirement that small changes by mutation occur more frequently than large ones, and a quantification of the selective pressure imposed by the most commonly used selection operators). Nevertheless, evolutionary algorithms often yield excellent results when applied to complex optimization problems where other methods are either not applicable or turn out be unsatisfactory (a variety of examples can be found in the Handbook of Evolutionary Computation).
Important practical problem classes where evolutionary algorithms yield solutions of high quality include engineering design applications involving continuous parameters (e.g., for the design of aircraft, structural mechanics problems based on two-dimensional shape representations, electromagnetic systems, and mobile manipulators) discrete parameters (e.g., for multiplierless digital filter optimization, the design of a linear collider, or nuclear reactor fuel arrangement optimization), and mixed-integer representations (e.g., for the design of survivable networks and optical multilayer systems).
Combinatorial optimization problems with a straightforward binary representation of solutions have also been treated successfully with canonical genetic algorithms and their derivatives (e.g., set partitioning and its application to airline crew scheduling, knapsack problems and others). Relevant applications to combinatorial problems utilizing a permutation representation of solutions are also found in the domains of scheduling (e.g., production scheduling and related problems), routing (e.g. of vehicles or telephone calls), and packing (e.g. of pallets on a truck).
The existing range of successful applications is extremely broad, thus by far preventing an exhaustive overview — the list of fields and example applications should be taken as a hint for further reading rather than a representative overview. Some of the most challenging applications with a large profit potential are found in the field of biochemical drug design, where evolutionary algorithms have gained remarkable interest and success in the past few years as an optimization procedure to support protein engineering. Also finance and business provide a promising field of profitable applications, but of course few details are published about this work. In fact, the relation between evolutionary algorithms and economics has found increasing interest in the past few years and is now widely seen as a promising modeling approach for agents acting in a complex, uncertain situation.

8 Summary

Evolutionary computation, as of today, is a wide and rapidly expanding field that has established itself as one of the key technologies for building adaptive systems which are able to deal with complex and changing environments just as natural evolution does. The mainstream algorithms of evolutionary computation, namely genetic algorithms, evolution strategies, evolutionary programming and genetic programming, have been presented in this contribution on a rather general level, but with an emphasis on illustrating conceptual and technical differences between them. The evolution of evolutionary computation has caused many new developments of recombining and mutating existing algorithms, and today the boundaries between the classical approaches are almost invisible.
Also on the level of theoretical approaches, the interaction between genetic algorithm and evolution strategy researchers in particular has fundamentally improved the understanding of working processes of evolutionary algorithms, such that relatively strong performance predictions are possible.
Thereotical developments in the field also have an impact on practical applications, which are now based on a mature insight into the algorithm's working principles. Numerous highly successful practical applications of evolutionary algorithms are reported in the literature (and even more proprietary work remains unreported), and it is quite obvious to see the increasing demand of industry for this powerful technology at the beginning of the new century.

References

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.]

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.]

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.]

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

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.]

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.]

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.]

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.]

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.]

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

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.]

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.]

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

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.]

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.]

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.]

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.]

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.]

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.]