Genetic Algorithms: Basic Ideas, Variants and Analysis

Sharapov R.R.
Institute of Mathematics and Mechanics (IMM)
Russia

1. Introduction

Genetic algorithms are wide class of global optimization methods. As well as neural networks and simulated annealing, genetic algorithms are an example of successful using of interdisciplinary approach in mathematics and computer science. Genetic algorithm simulates natural selection and evolution process, which are well studied in biology. In most cases, however, genetic algorithms are nothing else than probabilistic methods, which are based on principles of evolution. The idea of genetic algorithm appears first in 1967 in J. D. Bagley's thesis (Bagley, 1967). The theory and applicability was then strongly influenced by J. H. Holland, who can be considered as the pioneer of genetic algorithms (Holland, 1992). Since then, this field has witnessed a tremendous development.
There are many applications where genetic algorithms are used. Wide spectrum of problems from various branches of knowledge can be considered as optimization problem. This problem appears in economics and finances, cybernetics and process control, game theory, pattern recognition and image analysis, cluster analysis etc. Also genetic algorithm can be adapted for multicriterion optimization task for Pareto-optimal solutions search. But most popular applications of genetic algorithm are still neural networks learning and fuzzy knowledge base generation.
There are three ways in using genetic algorithms with neural networks:

1. Weight learning. Optimal net weights are found with genetic algorithm when conventional methods (e.g. backpropagation) are not applicable. It is suitable when continuous activation function of neuron (such as sigmoid) is used, so error function become multiextremal and conventional method can find only local minimum.
2. Architecture optimization. Genetic algorithm is used for finding optimal net architecture from some parameterized class of net architectures.
3. Learning procedure optimization. In this expensive but effective method genetic algorithm is used for finding optimization parameters of learning function (weight correction function). Usually this method is used with architecture optimization simultaneously.

Genetic fuzzy systems are other popular application of genetic algorithms. Fuzzy system design consists of several subtasks: rule base generation, tuning of membership function and tuning of scaling function. All this tasks can be considered as optimization problem, so genetic algorithm is applicable (Cordon et al., 2004).

Source: Vision Systems: Segmentation and Pattern Recognition, ISBN 987-3-902613-05-9,
Edited by: Goro Obinata and Ashish Dutta, pp.546, I-Tech, Vienna, Austria, June 2007

The optimization problem solved by genetic algorithms in general can be formulated as:

$$\max_{x \in X} f(x) \tag{1}$$

where X is search space, objective function f is total function in X, f: $X \rightarrow \mathbb{R}$. Some particular cases of this problem are well studied and solution methods are well known. For instance it is mentioned linear and convex programming problem. In general, however, this problem is very complex and non-solvable. It means that solution cannot be obtained in finite iteration steps.
We restrict problem (1) and consider case of compact and simple structure of set X, e.g. X is hypercube and it is known that f reach maximum inside X. In this case complexity of optimization task is depended from complexity of objective function f only. In common case f is non-smooth (non-differentiable) multiextremal function. Even through f is differentiable and conventional optimization methods e.g. gradient descent are applicable there are no guarantee that global optimum will be found.
There are two wide classes of optimization methods to solve global optimization problem: deterministic and stochastic. First obtain solution via almost complete search all over the X, so these methods are slow and non-efficient, but guarantee optimum finding. Also using of these methods requires some restrictions on objective function, so in several cases deterministic methods are not applicable. Second class is stochastic methods, which are faster and more efficient and universal than deterministic but has one essential shortcoming: maximization of objective function is not guarantee. Most of stochastic algorithms evaluate objective function in some random points of search space. Then sample of these points is processed and some pointes are saved for the next iteration.
As the practice shows in many instances it is acceptable to find not best but just well solution, so stochastic methods and genetic algorithms particularly are very effective.

2. Basic Ideas and Concepts

We consider optimization problem (1). Genetic algorithm does not work with problem (1) directly, but with coded version of them. Search space X is mapped into set of string S. Function $c(x)$: $X \rightarrow S$ is called coding function. Conversely, function $d(s)$: $S \rightarrow X$ is called decoding function and $c \circ d(s) = s$ should be done for any string s. In practice, coding functions, which have to be specified depending on the needs of the actual problem, are not necessarily bijective, so $d \circ c$ is not identical map over X, but it is over $D = d(S)$.

Usually, S is finite set of binary strings:

$$S = \{0,1\}^m \tag{2}$$

where m is obviously length of string. Generally simple binary code or Gray code is used. Note that S is finite, but X is commonly not. So, we quantize search space and algorithm finds solution approximately, but solution precision can be made as high as needed by increasing m.
Thus, we replace problem (1) with follows:

$$\max_{s \in S} f(s) \tag{3}$$

where under *f(s)* we imply *f(d(s))*.

Terminology particularly borrowed from natural genetic and evolution theory is commonly used in framework of genetic algorithms. Below we give some of most often used terms.

Member of set *S* is called *individual*. Individual in genetic algorithm is identified with *chromosome*. Information encoded in chromosome is called *genotype*. *Phenotype* is values of source task variables corresponding to genotype. In other words phenotype is decoded genotype. In simple genetic algorithm chromosomes are binary string of finite length. *Gene* is a bit of this string. *Allele* is value of gene, 0 or 1. *Population* is finite set of individuals. Objective function of optimization problem is called fitness *function*.

Fitness of individual is value of fitness function on phenotype corresponding individual. *Fitness of population* is aggregative characteristic of fitness of individuals. Fitness of best individual or average fitness of individuals is commonly used as population fitness in genetic algorithms.

In process of evolution one population is replaced by another and so on, thus we select individuals with best fitness. So in the mean each next generation (population) is fitter than it predecessors. Genetic algorithm produces maximal fitness population, so it solve maximization problem. Minimization problem obviously reduced to maximization problem.

In simple genetic algorithm size of population *n* and binary string length *m* is fixed and don't changes in process of evolution. We can write basic structure of simple genetic algorithm in the following way:

```
Compute initial population;
WHILE stopping condition not fulfilled DO BEGIN
        select individuals for reproduction;
        create offsprings by crossing individuals;
        eventually mutate some individuals;
        compute new generation;
END
```

As obvious from the above stated algorithm, the transition from one generation to the next consists of three basic components:

Selection: Mechanism for selecting individuals for reproduction according to their fitness.

Crossover: Method of merging the genetic information of two individuals. In many respects the effectiveness of crossover is depended on coding.

Mutation: In real evolution, the genetic material can by changed randomly by erroneous reproduction or other deformations of genes, e.g. by gamma radiation. In genetic algorithms, mutation realized as a random deformation of binary strings with a certain probability.

These components are called genetic operators. We consider these operators more detailed below.

Compared with conventional continuous optimization methods, such as gradient descent methods, we can state the following significant differences:

1. Genetic algorithms manipulate coded versions of the problem parameters instead of the parameters themselves, i.e. the search space is *S* instead of *X* itself. So, genetic algorithm finds solution approximately.
2. 2. While almost all conventional methods search from a single point, genetic algorithm always operates on a whole population of points (strings-individuals). It improves

robustness of algorithm and reduces the risk of becoming trapped in a local stationary point.

3. Normal genetic algorithms do not use any auxiliary information about the objective function value such as derivatives. Therefore, they can be applied to any kind of continuous or discrete optimization problem.
4. Genetic algorithms use probabilistic transition operators while conventional methods for continuous optimization apply deterministic transition operators. More specifically, the way a new generation is computed from the actual one has some random.

3. Simple genetic algorithm

Here we consider simpler genetic algorithm in more detail. As previously noted let m is binary string space dimension, n is population size. The generation at time t is a list of n binary strings, which we will denote with

$$B_t = (b_{1,t}, b_{2,t}, \ldots, b_{n,t}) \tag{4}$$

Stated above basic structure of genetic algorithm can be written more detailed in the following way:

```
$t$ := 0;
Compute initial population $B_0$;
WHILE stopping condition not fulfilled DO BEGIN
        FOR $i$:=1 TO $n$ DO
                select $b_{i,t+1}$ from $B_t$
        FOR $i$:=1 TO $n$ STEP 2 DO
                with probability $p_c$ perform crossover of $b_{i,t+1}$ and $b_{i+1,t+1}$
        FOR $i$:=1 TO $n$ DO
                with probability $p_m$ eventually mutate $b_{i,t+1}$
        $t$:=$t$+1;
END
```

We don't give concrete expression for stopping condition because these conditions have no features in comparison with other global optimization methods. So, as such conditions we can take restriction on number of iterations or some phenotype convergence conditions. Last can be formulated in terms of maximal or average fitness.

Commonly used procedure to compute initial population consist in random selection of n points uniformly distributed over the search space. If additional information about decision region is presented, it can be used for initial population computation.

3.1 Selection

Selection is the component which guides the algorithm to the solution by preferring individuals with high fitness over low-fitted ones. It realizes "The fittest will survive" principle. Selection can be a deterministic operation, but in most implementations it has random components.

One variant, which is very popular nowadays, is the following scheme, where the probability to choose a certain individual is proportional to its fitness. It can be regarded as a random experiment with

$$P\{b_{i,t}\} = p_{i,t} = \frac{f(b_{i,t})}{\sum_{k=1}^{n} f(b_{k,t})} \tag{5}$$

Of course, this formula only makes sense if all the fitness values are positive. If this is not the case, a increasing transformation φ: $\mathbb{R} \to \mathbb{R}^+$ must be applied. In simple case shift $\varphi = x+M$ can be used, where M is sufficiently great. M is chosen based upon some information about fitness function. If there no such information other transformations must be applied, such as exponential $\varphi = a^x$ or shifted arctangent $\varphi = arctan(x)+\pi/2$. Then the probabilities can be expressed as

$$P\{b_{i,t}\} = p_{i,t} = \frac{\varphi(f(b_{i,t}))}{\sum_{k=1}^{n} \varphi(f(b_{k,t}))} \tag{6}$$

Everywhere below we suppose that function *f* is positive.
We can force the property (5) to be satisfied by applying a random experiment which is, in some sense, a generalized roulette game. In this roulette game, the slots are not equally wide, i.e. the different outcomes can occur with different probabilities. Figure 1 gives a graphical interpretation of this roulette wheel game.

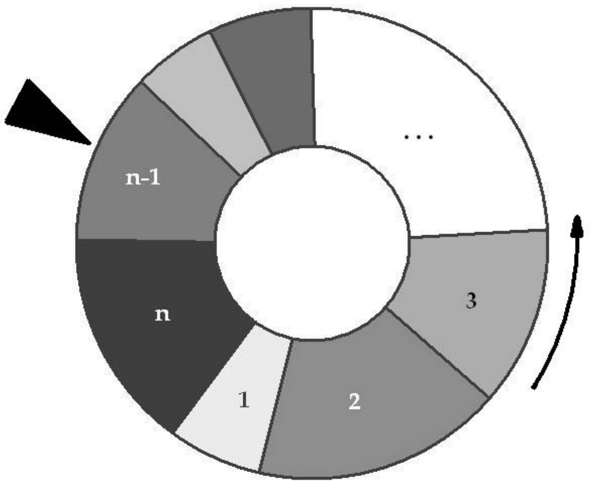

Fig. 1. A graphical representation of roulette wheel selection

For obvious reasons, this method is often called proportional selection. Mean of copies of individual $b_{i,t}$ which will be selected for follows crossover can be expressed as

$$\mathrm{E}(\textit{number of copies } b_{i,t}) = p_{i,t}n \tag{7}$$

It is easy to see that ill-fitted individuals have slim chance to leave offsprings, so they leave population very early. In some cases, this can be the cause of premature convergence of algorithm into local maxima. On the other hand, refinement in the end phase can be slow since the individuals have similar fitness values. These problems can be overcome by using alternative selection schemes:

Linear rank selection. Rank of the fitness as the basis of selection is used instead of the values themselves.

Tournament selection. In this scheme, a small group of individuals is sampled from the population and the individual with best fitness is chosen for reproduction. This selection scheme is also applicable when the fitness function is given in implicit form, i.e. when we only have a comparison relation which determines which of two given individuals is better.

3.2 Crossover

In sexual reproduction, as it appears in the real world, the genetic material of the two parents is mixed when the gametes of the parents merge. Usually, chromosomes are randomly split and merged, with the consequence that some genes of a child come from one parent while others come from the other parents.

This mechanism is called crossover. It is a very powerful tool for introducing new genetic material and maintaining genetic diversity, but with the outstanding property that good parents also produce well-performing children or even better ones.

Basically, crossover is the exchange of genes between the chromosomes of the two parents. In the simplest case, this process in genetic algorithms is realized by cutting two strings at a randomly chosen position (crossing point) and swapping the two tails. This process, which called one-point crossover, is visualized in Figure 2. In genetic algorithm selected individuals paired in some way and then crossing over with probability p_c.

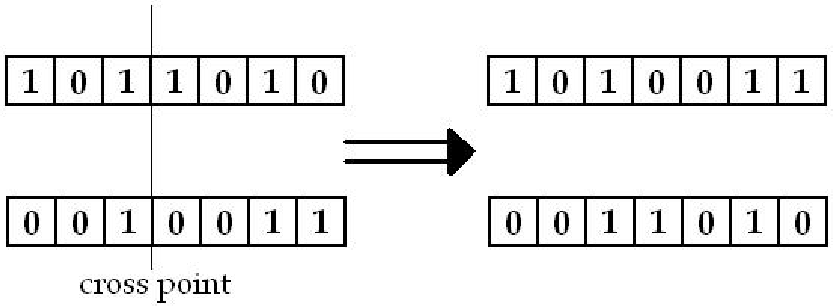

Fig. 2. One-point crossover of binary strings

One-point crossover is a simple and often-used method for genetic algorithms which operate on binary strings. For other problems or different coding function, other crossover methods can be useful or even necessary. We mention some of them, for more details see (Goldberg, 1989).

N-point crossover. Instead of only one, N breaking points are chosen randomly. Every second section is swapped. Among this class, two-point crossover is particularly important.

Segmented crossover. Similar to N-point crossover with the difference that the number of breaking points can vary.

Uniform crossover. For each position, it is decided randomly if the positions are swapped.

Shuffle crossover. First a randomly chosen permutation is applied to the two parents, then N-point crossover is applied to the shuffled parents, finally, the shuffled children are transformed back with the inverse permutation.

3.3 Mutation

Mutation is powerful factor of variability and consists in random deformation of genetic material. In real world these deformations take place as result of radioactivity, ultraviolet radiation or viruses influence. In real reproduction, the probability that a certain gene is mutated is almost equal for all genes. Mutation in genetic algorithm is analogue of natural one: each gene of chromosome is inverted with probability p_m, so this mutation is called uniform mutation. Also, in genetic algorithms alternative mutation methods are used. We mention some of them, more detailed see (Goldberg, 1989).
Inversion of single bits. With probability p_m, one randomly chosen bit is negated.
Bitwise inversion. The whole string is inverted bit by bit with probability p_m.
Random mutation. With probability p_m, the string is replaced by a randomly chosen one.

4. Variants

We consider simple variant of genetic algorithm, but it is sufficiently effective. Thus, there are some ways to improve efficiency and robustness. In this section we consider some of this ways.
Elitism is very effective element that realizes "best must survive" principle. It can be added into any selection scheme and consists in follows: best individual from parent population is compared with best individual from offspring population and best of them is added into next generation. Elitism guarantees that next generation fitness will be better or equal than parent generation fitness. Elitism is often-used element, but it should, however, be used with caution, because it can lead to premature convergence.
Adaptive genetic algorithms are algorithms whose parameters, such as the population size, the crossing over probability, or the mutation probability are varied while the genetic algorithm is running. A simple variant could be the following: The mutation rate is changed according to changes in the population; the longer the population does not improve, the higher the mutation rate is chosen. Vice versa, it is decreased again as soon as an improvement of the population occurs.
Hybrid genetic algorithms are used when additional auxiliary information such as derivatives or other specific knowledge is known about objective function. So, conventional method, such as gradient descent is applicable. The basic idea is to divide the optimization task into two complementary parts. The coarse, global optimization is done by the genetic algorithm while local refinement is done by the conventional method. A number of variants is reasonable:

1. The genetic algorithm performs coarse search first. After it is completed, local refinement is done.
2. The local method is integrated in the genetic algorithm. For instance, every k generations, the population is doped with a locally optimal individual.
3. Both methods run in parallel: All individuals are continuously used as initial values for the local method. The locally optimized individuals are re-implanted into the current generation.

In *self-organizations genetic algorithms* not only data is object of evolution. Parameters of genetic algorithm, such as coding function or genetic operator parameters, are optimized too. If this is done properly, the genetic algorithm could find its own optimal way for representing and manipulating data automatically.

5. Analysis

As stated above, genetic algorithm is stochastic optimization method and not guarantees convergence to solution. Therefore, we consider convergence in terms of mean. Convergence analysis becomes complicated by using three stochastic operators: selection, crossover and mutation that have many variations, so there are many different algorithms. We consider simple genetic algorithm with fixed population size n operates in space of binary string with fixed length m. It is assumed that one-point crossover, uniform mutation and proportional selection are used.

5.1 The Schema Theorem

Analysis of genetic algorithm we start from classic result of Holland – the so-called Schema Theorem. But at first we'll make some definitions.

Definition 1. A string $H = h_1 \ldots h_m$ over the alphabet {0, 1, *} is called a (binary) *schema* of length m. An $h_i = 0$ or 1 is called a *specification* of H, an $h_i = *$ is called *wildcard*. Schemata can be considered as specific subsets of $\{0, 1\}^m$.

If we interpret binary strings space as hypercube with dimension m, then schemata can be interpreted as hyperplanes (see Figure 3).

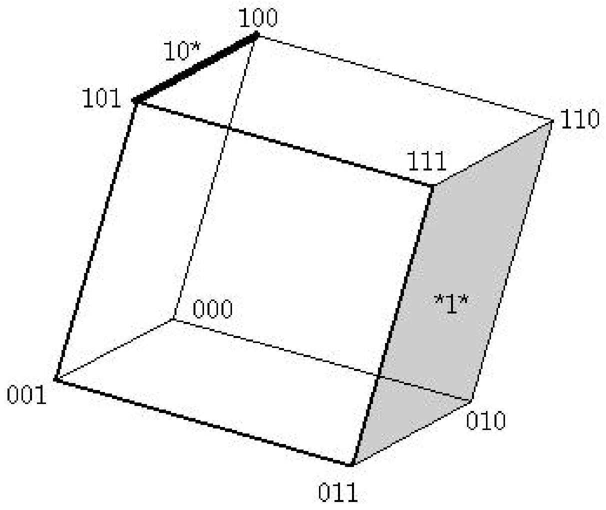

Fig. 3. Schemata as hyperplanes in hypercube

Obviously number of schemata is 3^m.

Definition 2. A string $S = s_1 \ldots s_m$ over the alphabet {0, 1} *fulfills* the schema $H = h_1 \ldots h_m$ if and only if it matches H is all non-wildcard positions:

$$\forall i \in \{ j \mid h_j \neq * \} : s_i = h_i \tag{8}$$

Definition 3. The number of specifications of a schema H is called *order* and denoted as

$$O(H) = |\{ i \mid 1 \leq i \leq m, h_i \neq * \}| \tag{9}$$

Definition 4. The distance between the first and the last specification

$$\delta(H) = \max\{i \mid h_i \neq *\} - \min\{i \mid h_i \neq *\} \tag{10}$$

is called the *defining length* of a schema H.
Also let us make some notations:
The number of individuals which fulfill H at time step t are denoted as $r_{H,t}$.
The observed average fitness at time t is denoted as:

$$\overline{f}(t) = \frac{1}{n}\sum_{k=1}^{n} f(b_{k,t}) \tag{11}$$

The observed average fitness of schema H in time step t is denoted as:

$$\overline{f}(H,t) = \frac{1}{r_{H,t}} \Sigma\{f(b_{i,t}) \mid b_{i,t}\, \mathit{fulfill}\ H\} \tag{12}$$

The following theorem holds.
Theorem (Schema Theorem—Holland 1975).
The following inequality holds for every schema H:

$$\mathrm{E}(r_{H,t+1}) \geq r_{H,t}\,\frac{\overline{f}(H,t)}{\overline{f}(t)}(1-p_c\,\frac{\delta(H)}{m-1})(1-p_m)^{O(H)} \tag{13}$$

where E is mean of number of next generation individuals fulfills schema H. More generally statement of schema theorem can be formulated as follows:

$$\mathrm{E}(r_{H,t+1}) \geq r_{H,t}\,\frac{\overline{f}(H,t)}{\overline{f}(t)}\,P_c(H)\,P_m(H) \tag{14}$$

where estimations P_c and P_m depend only from schema H on one hand and crossover and mutation methods correspondingly on another. Such estimations can be obtained for all considered variants of crossover and mutation operators. One can see (Holland, 1992) for full proof of schema theorem.
The schema theorem answer the question what schemata has more chance to survive, but say nothing about convergence in essence.

5.2 Building blocks hypothesis

As obviously follow from schema theorem high-fitness schemata with low order and short length have more chance to survive in process of evolution. Let p_m is sufficiently small, then (13) takes the form:

$$\mathrm{E}(r_{H,t+1}) \geq r_{H,t}\,\frac{\overline{f}(H,t)}{\overline{f}(t)}(1-p_c\,\frac{\delta(H)}{m-1})(1-p_m O(H)) \tag{15}$$

If population size is sufficiently great, then deviations from average $\mathrm{E}(r_{H,t+1})$ are very small. If we disregard them follows statement take place:

$$r_{H,t+1} \geq r_{H,t}\,\frac{\overline{f}(H,t)}{\overline{f}(t)}(1-p_c\,\frac{\delta(H)}{m-1})(1-p_m O(H)) \tag{16}$$

It is obviously follows from this recurrent expression that number of individuals fulfills high-fitness schemata with low $\delta(H)$ and $O(H)$ exponentially grows in process of evolution. Such schemata, i.e. well-fitted schemata with short length and low order, are called *building blocks*. Goldberg conjecture follows: *A genetic algorithm creates stepwise better solutions by recombining, crossing, and mutating short, low-order, high-fitness schemata.* This conjecture is called *building blocks hypothesis* (Goldberg, 1989).

If building blocks hypothesis is true, key role for convergence play coding method. Coding must be realized building blocks hypothesis concept. For example consider two examples of fitness function. First is an affine linear fitness function:

$$f(s) = a + \sum_{i=1}^{m} c_i s_i \tag{17}$$

where s_i is ith allele of chromosome s.

Second function correspond "needle-in-haystack" problem:

$$f(x) = \begin{cases} 1, x = x \\ 0, x \neq x_0 \end{cases} \tag{18}$$

In the linear case, the building block hypothesis seems justified, i.e. the fitness is computed as a linear combination of all genes. It is easy to see that the optimal value can be determined for every gene independently. For the second function, however, it cannot be true, since there is absolutely no information available which could guide a genetic algorithm to the global solution through partial, sub-optimal solutions. In other words, the more the positions can be judged independently, the easier it is for a genetic algorithm. On the other hand, the more positions are coupled, the more difficult it is for a genetic algorithm (and for any other optimization method). There is a special term derived from biology for this phenomena – *epistasis*. High *epistatic* problem are very difficult to solve. Genetic algorithms are appropriate for medium *epistatic* problems, and low *epistatic* problem can be solved much more efficiently with conventional methods.

Follow question may arise after analysis: what a genetic algorithm really processes, strings or schemata? The answer is both. Nowadays, the common interpretation is that a genetic algorithm processes an enormous amount of schemata implicitly and simultaneously. This is accomplished by exploiting the currently available, incomplete information about these schemata continuously, while trying to explore more information about them and other, possibly better schemata.

5.3 The Convergence Theorem

The Schema Theorem clarifies some aspects of the mechanism how genetic algorithm works. Building blocks hypothesis conjecture some assumption about convergence, but it isn't proven. Some results about convergence were obtained by author. Although genetic algorithm not guarantees solution finding, it converge in the mean. Below we formulate Theorem of Convergence of genetic algorithms.

As stated above, we consider simple genetic algorithm with fixed population size n operates in space of binary string with fixed length m. It is assumed that one-point crossover with probability p_c, uniform mutation with probability p_m and proportional selection are used.

Also we assume that elitism is incorporated in selection procedure, so best individual always survive. Hence, the following theorem holds:
Theorem.
Let $p_m \leq 0.5$, and $S = (1 - p_m^m)\ (2 - (1 - p_c)^n) < 1$.
Then,

$$\lim_{k\to\infty} \mathrm{E}(f(B_k)) = f^* \tag{19}$$

where B_k is the population after the kth iteration step of the genetic algorithm, $f(B_k)$ is the maximal fitness over the population B_k, and f^* is the required optimal value. $\mathrm{E}(f(B_k))$converges to f^* non-decreasingly. Proof of this theorem can be found in (Sharapov & Lapshin, 2006).
There is an interesting corollary corresponding case of zero p_c.
Corollary. Let $p_m \leq 0.5$, $p_c = 0$.
Then,

$$\lim_{k\to\infty} \mathrm{E}(f(B_k)) = f^* \tag{20}$$

where B_k is the population after the kth iteration step of the genetic algorithm, $f(B_k)$ is the maximal fitness over the population B_k, and f^* is the required optimal value and $C_m^{[m/2]}$ is binomial coefficient. $\mathrm{E}(f(B_k))$converges to f^* non-decreasingly. Evidently, crossover absence gives us everywhere convergent algorithm.

6. Real-coded evolutionary optimization methods

Most of optimization problems have real-valued parameters (i.e. X is subset of $\mathbb{R}^N$, where N is problem dimension). It is clear that discretization approach applied in simple genetic algorithm has several shortcomings:

1. Continuum set of possible values is reduced to finite set of binary strings. So we limit considered search space, and if solution of task is located outside considered region, we will not find it.
2. The accuracy of the solution is limited by the discretization width $1/(2m-1)$, where m is length of binary string. Although precision can be improved by increasing m, it will require more computer power and time. Computational complexity grows exponentially with m growth.
3. It is complicated to choose appropriate coding method. Most often, no reasonable building blocks exist.

For these reasons, variants of genetic algorithms which are especially adapted to real-valued optimization problems have been proposed.

6.1 Real-coded genetic algorithms

Structure of real-coded genetic algorithm is not to differ from one considered in section 3. But chromosomes in real-coded genetic algorithms are represented as N-dimensional vectors of real numbers, where N is dimension of optimization problem:

$$b = (x_1, ..., x_N) \tag{21}$$

All selection schemes are applicable without any modifications. Crossover and mutation must be adapted.
In real-coded genetic algorithms follows crossover operators are used most-often:
Flat crossover. Two parents $b_1 = (x_{1,1}, \ldots, x_{1,N})$ and $b_2 = (x_{2,1}, \ldots, x_{2,N})$ are given, a vector of random values from the unit interval $\lambda = (\lambda_1, \ldots, \lambda_N)$ is chosen. The offspring $b' = (x'_1, \ldots, x'_N)$ is computed as a vector of linear combinations in the following way (for all $i = 1, \ldots, N$):

$$b'_i = \lambda_i x_{1,i} + (1 - \lambda_i) x_{2,i} \tag{22}$$

Second offspring from pair is computed analogously.
BLX-α crossover (Herrera et al., 1998) is an extension of flat crossover which allows an offspring allele x'_i to be also located outside the interval $[\min(x_{1,i}, x_{2,i}), \max(x_{1,i}, x_{2,i})]$. In BLX-α crossover, each offspring allele x'_i is chosen as a uniformly distributed random value from the interval

$$[\min(x_{1,i}, x_{2,i}) - I \cdot a, \max(x_{1,i}, x_{2,i}) + I \cdot a] \tag{23}$$

where $I = \max(x_{1,i}, x_{2,i}) - \min(x_{1,i}, x_{2,i})$.
The parameter α has to be chosen in advance. For α = 0, BLX-α crossover becomes identical to flat crossover.
Simple and discrete crossover is analogous to considered above classical one-point and uniform crossover.
The following mutation operators are most common for real-coded genetic algorithms:
Random mutation. For a randomly chosen gene i of an individual $b = (x_1, \ldots, x_N)$, the allele x_i is replaced by a randomly chosen value from a predefined interval $[a_i, b_i]$.
Non-uniform mutation. In non-uniform mutation, the possible impact of mutation decreases with the number of generations (Michalewicz, 1996). Assume that $t_{\max}$ is the predefined maximum number of generations. Then, with the same setup as in random mutation, the allele x_i is replaced by one of the two values

$$\begin{aligned} x'_i &= x_i + \Delta(t, b_i - x_i) \\ x''_i &= x_i - \Delta(t, x_i - a_i) \end{aligned} \tag{24}$$

The choice which of the two is taken is determined by a random experiment with two outcomes that have equal probabilities 0.5 and 0.5. The random variable $\Delta(t, x)$ determines a mutation step from the range $[0, x]$ in the following way:

$$\Delta(t, x) = x \cdot (1 - \lambda^{(1 - t/t_{\max})^r}) \tag{25}$$

In this formula, λ is a uniformly distributed random value from the unit interval. The parameter r determines the influence of the generation index t on the distribution of mutation step sizes over the interval $[0, x]$.

6.2 Evolutionary strategies

Evolutionary strategies are real-coded global optimization methods were developed in late 1960s mainly by I. Rechenberg independently from Holland's work on genetic algorithms. Chromosome in evolutionary strategies is represented by $2N$ dimensional vector, where N is dimension of problem:

$$b = (x_1, \ldots, x_N; \sigma_1, \ldots, \sigma_N) \tag{26}$$

The first half (x_1, …, x_N) corresponds to the potential solution of the optimization problem like in real-coded genetic algorithms. The second half (σ_1, …, σ_N) defines the vector of standard deviations for the mutation operation.

As usual, there are two means of modifying genetic material in evolutionary strategies: a recombination operation that could be understood as some kind of crossover and mutation. Unlike genetic algorithms, mutation plays a more central role in evolutionary strategies. Usually as recombination operator flat or discrete crossover applied in real-coded genetic algorithm (see previous section) are used. Most often in evolutionary strategies flat recombination with $\lambda = 0.5$ is used (so-called intermediate recombination).

Mutation in evolutionary strategies consists of two phases. Firstly, normal distributed noise is added to each allele x_i. More specifically, for all $i = 1, \ldots, N$, the mutated allele is given as

$$x'_i = x_i + N(0, \sigma_i^2) \tag{27}$$

where N(0,σ^2_i) is normally distributed random variable with zero mean and standard deviation σ_i.

Secondly, we added logarithmically normal distributed noise to σ_i alleles:

$$\sigma'_i = \sigma_i \cdot \exp(\tau' N(0,1) + \tau N_i(0,1)) \tag{28}$$

The factor exp (τ' N(0,1)) is an overall factor increasing or decreasing the "mutability" of the individual under consideration. Note that N(0,1) is chosen only once for the whole individual when it is mutated. The factor exp (τ N_i(0,1)) locally adapts the mutation step sizes. Note that, in this second factor, the normally distributed random value N_i(0,1) is chosen separately for each gene. The adaptation of mutation step sizes in evolutionary strategies has the particular advantage that no parameters have to be chosen in advance. Instead, they evolve during the run of an evolutionary strategy in a self-organizing way.

The two parameters τ and τ' have to be chosen in advance. Schwefel has proposed to choose these parameters in the following way (Schwefel, 1995):

$$\tau' = \frac{1}{\sqrt{2N}}, \tau = \frac{1}{\sqrt{2\sqrt{N}}} \tag{29}$$

Selection in evolutionary strategies also has some features in comparison with genetic algorithms. The nowadays commonly accepted selection and sampling schemes in evolutionary strategies are the following:

(μ + λ)-strategy: a number of μ parents are selected from the current generation. These μ parents are used to generate a number of λ offsprings, which have been generated by some recombination and/or mutation operations. Out of the union of parents and offsprings (in total, a number of $\mu + \lambda$), the best μ are kept for the next generation. Note that the (μ + λ)-strategy inherently incorporates elitism.

(μ, λ)-strategy: in this scheme, which is nowadays considered the standard selection/sampling strategy in evolutionary strategies, again μ parents are selected from the current generation and used to generate λ offsprings (with the additional restriction $\lambda \geq \mu$). The parents are discarded completely and the best μ offsprings are kept for the next generation. The (μ, λ)-strategy does not incorporate elitism.

Note that both strategies only use the ranking of fitness values. Therefore, they can be applied both to minimization and maximization problems, without any need for scaling or transforming fitness values.

6.3 Evolutionary programming

Idea of evolutionary programming were proposed by L.J.Fogel in the middle of 1960s and later extended by his son D.B. Fogel (Fogel, 1992). Evolutionary programming solves same tasks in similar ways as real-coded genetic algorithms and evolutionary strategies. An important difference evolutionary programming from real-coded genetic algorithms and evolutionary strategies is consists in following: evolutionary programming does not use crossover or any other kind of exchange of genetic material between individuals. Offsprings are generated by mutation only.

We consider modified evolutionary programming method (Fogel, 1992). As well as evolutionary strategies, in this variant of evolutionary programming individual is represented by $2N$ dimensional vector of real values, where N is dimension of problem:

$$b = (x_1, \ldots, x_N; v_1, \ldots, v_N) \tag{30}$$

The second half of the vector (v_1, …, v_N) contains the variances of the mutation step sizes, as the mutation is done in the following way:

$$\begin{aligned} x'_i &= x_i + \sqrt{v_i} \cdot N_i(0,1) \\ v'_i &= v_i + \sqrt{\chi v_i} \cdot N_i(0,1) \end{aligned} \tag{31}$$

Unfortunately, it is not guaranteed that v'_i is positive. Therefore, additional measures have to be taken to avoid that v'_i gets 0 or negative. The parameter χ defines volatility of mutation factors v_i. $N_i(0,1)$ is a value of standard normally distributed random variable which is chosen separately for each gene.

Evolutionary programming uses a kind of combination of tournament and linear rank selection. The fitness of an individual b is compared with q other randomly picked competitors taken from the union of μ parents and λ offsprings. The score w_i of the individual b is computed as the number of individuals within the q selected ones that have a lower fitness than b. The parents and offsprings are ranked according to their score and the best μ are selected for the next generation. Note that this selection scheme inherently incorporates elitism. Moreover, for large q, it behaves almost in the same way as the ($\mu + \lambda$)-strategy used in evolutionary strategies.

6.4 Analysis of convergence of real-coded methods

We will not investigate convergence detailed here and will make only some assertions about convergence properties.

For simplicity we consider the case of evolutionary programming only (see previous section), where $N = 1$. Also let $\lambda = \mu$ is used in selection scheme. Let $B_0 = (b_{0,1}, \ldots, b_{0,\mu})$ is initial population. Individuals in population are descending sorted by fitness, so first individual is best of all. After mutation we obtain μ offsprings b'_1, …, b'_μ. Each of them is two-dimensional normally distributed variate. Since genes mutate independently they are independent variates. Then consider aggregate population of parents and offsprings:

$$B' = (b_{0,1},\dots,b_{0,\mu}; b'_1,\dots,b'_\mu) \tag{32}$$

Best individual from this population is kept for the next generation, because its rank is maximal among all of them. Let z – best among offsprings $b'_1\dots b'_\mu$. Obviously z is two-dimensional random variable. Then best individual $b_{1,1}$ of next generation B_1 is best of $b_{0,1}$ and z, i.e. $b_{1,1}$ = arg max $\{b_{0,1}, z\}$. So

$$f(B_1) = f(b_{1,1}) = \max\{f(b_{0,1}), f(z)\} \tag{33}$$

Assume, that $f(z)$ is absolutely continuous random variate. Let's consider mean of variate $f(b_{1,1})$ (here and below we suppose that all integrals are exist and converge absolutely):

$$\mathrm{E}f(B_1) = \mathrm{E}f(b_{1,1}) = \mathrm{E}\max\{f(b_{0,1}), f(z)\} = \int_{-\infty}^{+\infty}\max\{f(b_{0,1}), f(z)\}d(x)dx \tag{34}$$

where $d(x)$ is density function of variate $f(z)$. Transom (34):

$$\mathrm{E}f(B_1) = f(b_{0,1})\int_{-\infty}^{f(b_{0,1})} d(x)dx + \int_{f(b_{0,1})}^{+\infty} xd(x)dx = f(b_{0,1}) + \int_{f(b_{0,1})}^{+\infty}(x - f(b_{0,1}))d(x)dx \tag{35}$$

Subintegral expression of second item is obviously non-negative. Therefore

$$\int_{f(b_{0,1})}^{+\infty}(x - f(b_{0,1}))d(x)dx \ge 0 \tag{36}$$

Consider the case of equality more detailed. Obviously equality is realized if and only if subintegral expression is identically zero, so $d(x) \equiv 0$ on interval $(f(b_{0,1}), +\infty)$. Hence probability

$$P\{f(z) > f(b_{0,1})\} = \int_{f(b_{0,1})}^{+\infty} d(x)dx = 0 \tag{37}$$

It means that improvement of fitness of population is impossible event. Since function f is continuous and genes are independent normally distributed variates, it is possible only if range of function f and interval $(f(b_{0,1}), +\infty)$ has no intersections (it could be verified if inverse assumption was made), so $f(b_{0,1})$ is a global maxima of function f.
Thus, we obtain that either $\mathrm{E}f(B_1)=f(B_0)$ and solution is found or $\mathrm{E}f(B_1) > f(B_0)$. This deduction can be made for any step of algorithm, so following assertion holds:
If solution is not found on kth step of evolutionary programming algorithm, then

$$\mathrm{E}f(B_{k+1}) > f(B_k) \tag{38}$$

7. Concluding remarks

We consider simple genetic algorithm and some of variants. Also we have collected several important results which provide valuable insight into the intrinsic principles of genetic algorithms. Finally we consider real-valued optimization problem and some evolutionary method to solve it. Several remarks were made about convergence one of them. But mainly we consider genetic algorithms in itself. The future of this method, however, is in union with neural networks and fuzzy systems. Below, we mention some perspective approaches:

1. Fuzzy genetic programming. Genetic programming is concerned with the automatic generation of computer programs. Fuzzy genetic programming combines a simple genetic algorithm that on a context-free language with a context-free fuzzy rule language.
2. Genetic fuzzy systems. As mentioned in introduction of this chapter these systems use evolutionary methods for rule base generation and tuning.
3. Genetic fuzzy neural networks. Genetic fuzzy neural networks are the result of adding genetic or evolutionary learning capabilities to systems integrating fuzzy and neural concepts. The usual approach of most genetic fuzzy neural networks is that of adding evolutionary learning capabilities to a fuzzy neural network.
4. Genetic fuzzy clustering algorithm. Genetic algorithms can be used in fuzzy clustering. Most widely used method is to optimize parameters of so-called C-mean FCM-type algorithms, that can improve it performance. Another approach is based on directly solving the fuzzy clustering problem without interaction with any FCM-type algorithm.

8. References

Bagley, J. D. (1967). The Behavior of Adaptive Systems Which Employ Genetic and Correlative Algorithms. PhD thesis, University of Michigan, Ann Arbor.

Cordon, O; F. Gomide, F.; Herrera, F.; Hoffmann, F. & Magdalena L. (2004). Ten years of genetic fuzzy systems: current framework and new trends, *Fuzzy sets and systems*, No. 141, pp. 5-31, ISSN 0165-0114.

Fogel, D. B. (1992). *Evolving Artificial Intelligence.* PhD thesis, University of California, San Diego.

Goldberg, D. E. (1989). *Genetic Algorithms in Search, Optimization, and Machine Learning,* Addison-Wesley, ISBN 0201157675, Reading, MA.

Herrera, F.; Lozano, M. & Verdegay, J. L. (1998). Tackling real-coded genetic algorithms: Operators and tools for behavioural analysis, *Artificial Intelligence Review*, Vol. 12, pp. 265–319. ISSN 0269-2821.

Holland, J. H. (1992). *Adaptation in Natural and Artificial Systems,* MIT Press, ISBN 0-262-58111-6, Cambridge, MA.

Michalewicz, Z. (1996). *Genetic Algorithms + Data Structures = Evolution Programs*, 3rd extended ed., ISBN 3540606769, Springer, Heidelberg.

Schwefel, H.-P. (1995). Evolution and Optimum Seeking. Sixth-Generation Computer Technology, Series. John Wiley & Sons, New York.

Sharapov, R. R. & Lapshin, A.V. (2006). Convergence of Genetic Algorithms, *Pattern Recognition and Image Analysis,* Vol. 16, No. 3, pp.392-397. ISSN 1054-6618.