131: Evolutionary Computation

Ajith Abraham
Oklahoma State University, Stillwater, OK, USA

1 INTRODUCTION AND BIOLOGICAL MOTIVATION

A general introduction to artificial intelligence methods of measurement signal processing is given in **Article 128, Nature and Scope of AI Techniques, Volume 2**.

In nature, evolution is mostly determined by natural selection or different individuals competing for resources in the environment. Those individuals that are better are more likely to survive and propagate their genetic material. The encoding for genetic information (genome) is done in a way that admits asexual reproduction, which results in offspring that are genetically identical to the parent. Sexual reproduction allows some exchange and reordering of chromosomes, producing offspring that contain a combination of information from each parent. This is the recombination operation, which is often referred to as *crossover* because of the way strands of chromosomes cross over during the exchange. The diversity in the population is achieved by mutation operation.

Evolutionary algorithms are ubiquitous nowadays, having been successfully applied to numerous problems from different domains, including optimization, automatic programming, signal processing, bioinformatics, social systems, and so on. In many cases, the mathematical function, which describes the problem, is not known, and the values at certain parameters are obtained from simulations. In contrast to many other optimization techniques, an important advantage of evolutionary algorithms is they can cope with multimodal functions.

Usually found grouped under the term evolutionary computation or evolutionary algorithms (Bäck, 1996), are the domains of genetic algorithms (GA) (Holland, 1975), evolution strategies (Rechenberg, 1973; Schwefel, 1977), evolutionary programming (Fogel, Owens and Walsh, 1966), and genetic programming (Koza, 1992).

These all share a common conceptual base of simulating the evolution of *individual* structures via processes of *selection*, recombination, and mutation *reproduction*, thereby producing better solutions. The processes depend on the perceived performance of the individual structures as defined by the problem.

A population of candidate solutions (for the optimization task to be solved) is initialized. New solutions are created by applying reproduction operators (crossover and/or mutation). The fitness (how good the solutions are) of the resulting solutions is evaluated and suitable selection strategy is then applied to determine which solutions will be maintained into the next generation. The procedure is then iterated, as illustrated in Figure 1.

Handbook of Measuring System Design, edited by Peter H. Sydenham and Richard Thorn.
© 2005 John Wiley & Sons, Ltd. ISBN: 0-470-02143-8.

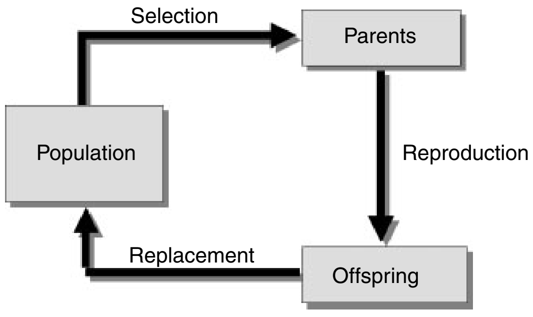

Figure 1. Flow chart of an evolutionary algorithm.

A primary advantage of evolutionary computation is that it is conceptually simple.

The procedure may be written as the difference equation:

$$x(t+1) = s\{v[x(t)]\} \tag{1}$$

where $x(t)$ is the population at time t under a representation x, v is a random variation (reproduction) operator, and s is the selection operator (Fogel, 1999).

1.1 Advantages of evolutionary algorithms

Following are some of the advantages of using evolutionary algorithms while compared to other global optimization techniques (Fogel, 1999).

1. Evolutionary algorithm performance is representation independent in contrast to other numerical techniques, which might be applicable for only continuous values or other constrained sets.
2. Evolutionary algorithms offer a framework such that it is comparably easy to incorporate prior knowledge about the problem. Incorporating such information focuses the evolutionary search, yielding a more efficient exploration of the state space of possible solutions.
3. Evolutionary algorithms can also be combined with more traditional optimization techniques. This may be as simple as the use of a gradient minimization after primary search with an evolutionary algorithm (e.g. fine tuning of weights of an evolutionary neural network) or it may involve simultaneous application of other algorithms (e.g. hybridizing with simulated annealing or Tabu search to improve the efficiency of basic evolutionary search).
4. The evaluation of each solution can be handled in parallel and only selection (which requires at least pair-wise competition) requires some serial processing. Implicit parallelism is not possible in many global optimization algorithms like simulated annealing and Tabu search.
5. Traditional methods of optimization are not robust to the dynamic changes in the problem of the environment and often require a complete restart in order to provide a solution (e.g. dynamic programming). In contrast, evolutionary algorithms can be used to adapt solutions to changing circumstance.
6. Perhaps, the greatest advantage of evolutionary algorithms comes from the ability to address problems for which there are no human experts. Although human expertise should be used when it is available, it often proves less than adequate for automating problem-solving routines.

2 GENETIC ALGORITHMS

A typical flowchart of a genetic algorithm is depicted in Figure 2. One iteration of the algorithm is referred to as a *generation*. The basic GA is very generic, and there are many aspects that can be implemented differently according to the problem (e.g. representation of solution (chromosomes), type of encoding, selection strategy, type of crossover and mutation operators, etc.). In practice, GAs are implemented by having arrays of bits or characters to represent the chromosomes. The individuals in the population

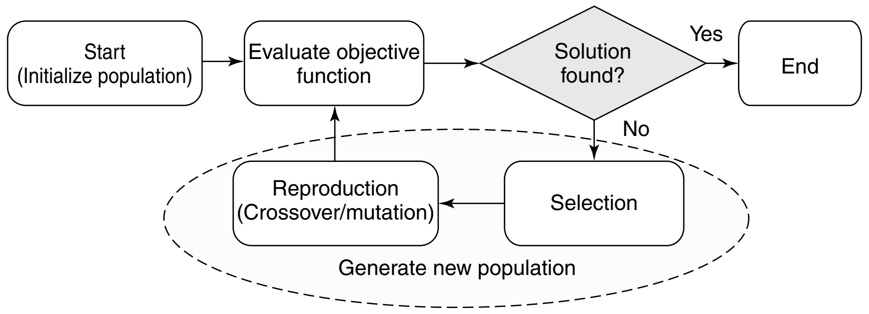

Figure 2. Flow chart of basic genetic algorithm iteration.

then go through a process of simulated evolution. Simple bit manipulation operations allow the implementation of crossover, mutation, and other operations. The number of bits for every gene (parameter) and the decimal range in which they decode are usually the same, but nothing precludes the utilization of a different number of bits or range for every gene.

When compared to other evolutionary algorithms, one of the most important GA feature is its focus on fixed-length character strings, although variable-length strings and other structures have been used.

2.1 Encoding and decoding

In a typical application of GAs, the given problem is transformed into a set of genetic characteristics (parameters to be optimized) that will survive in the best possible manner in the environment. For example, optimizing a function

$$\min f(x_1, x_2) = (x_1 - 5)^2 + (x_2 - 2)^2 \text{for} - 3 \leq x_1 3 \leq; \\ - 8 \leq x_2 8 \leq \quad (2)$$

The parameters of the search are identified as x_1 and x_2, which are called the *phenotypes* in evolutionary algorithms. In genetic algorithms, the *phenotypes* (parameters) are usually converted to *genotypes* by using a coding procedure. Knowing the ranges of x_1 and x_2, each variable is to be represented using a suitable binary string. This representation using binary coding makes the parametric space independent of the type of variables used. The genotype (chromosome) should in some way contain information about solution, which is also known as *encoding*. GAs use a binary string encoding, as shown below.

Chromosome A:	110110111110100110110
Chromosome B:	110111101010100011110

Each bit in the chromosome strings can represent some characteristic of the solution. There are several types of encoding (e.g. direct integer or real numbers encoding), which directly depends on the problem.

Permutation encoding can be used in ordering problems, such as the traveling salesman problem (TSP) or task-ordering problem. In permutation encoding, every chromosome is a string of numbers, which represents numbers in a sequence. A chromosome using permutation encoding for a 9-city TSP problem will appear as follows:

Chromosome A:	4 5 3 2 6 1 7 8 9
Chromosome B:	8 5 6 7 2 3 1 4 9

The chromosome represents the order in which the salesman will visit the cities. Special care is taken to ensure that the strings represent real sequences after crossover and mutation. Floating-point representation is very useful for numeric optimization (e.g. for encoding the weights of a neural network).

It should be noted that in many recent applications, more sophisticated genotypes are appearing (e.g. chromosome can be a tree of symbols or a combination of a string and a tree, some parts of the chromosome are not allowed to evolve, etc.).

3 SCHEMA THEOREM

Theoretical foundations of evolutionary algorithms can be partially explained by the schema theorem (Holland, 1975), which relies on the concept of schemata. Schemata are templates that partially specify a solution (more strictly, a solution in the genotype space). If genotypes are strings built using symbols from an alphabet A, schemata are strings whose symbols belong to $A \cup (*)$. This extra-symbol (*) must be interpreted as a wildcard, being loci occupied by it, called *undefined*. A chromosome is said to match a schema if they agree in the defined positions.

For example, the string 10011010 matches the schemata 1******* and **011*** among others but does not match *1*11*** because they differ in the second gene (the first defined gene in the schema).

A schema can be viewed as a hyperplane in a k-dimensional space, representing a set of solutions with common properties. Obviously, the numbers of solutions that match a schema H depend on the number of defined positions in it. Another related concept is the *defining-length* of a schema, defined as the distance between the first and the last defined positions in it.

The GA works by allocating strings to best schemata exponentially through successive generations, this being the selection mechanism mainly responsible for this behavior. On the other hand, the crossover operator is responsible for exploring new combinations of the present schemata in order to get the fittest individuals. Finally, the purpose of the mutation operator is to introduce fresh genotypic material in the population.

4 SELECTION AND REPRODUCTION

Individuals for producing offspring are chosen using a selection strategy after evaluating the fitness value of each individual in the selection pool. Each individual in the selection pool receives a reproduction probability depending on its own fitness value and the fitness value of all other individuals in the selection pool. This fitness is used for the actual selection step afterwards. Some of the popular selection schemes are discussed below.

4.1 Roulette-wheel selection

The simplest selection scheme is the roulette-wheel selection, also called *stochastic sampling with replacement*. This technique is analogous to a roulette wheel with each slice proportional in size to the fitness. The individuals are mapped to contiguous segments of a line such that each individual's segment is equal in size to its fitness. A random number is generated and the individual whose segment spans the random number is selected. The process is repeated until the desired number of individuals is obtained. As illustrated in Figure 3, chromosome 1 has the highest probability for being selected since it has the highest fitness.

4.2 Rank-based fitness assignment

In rank-based fitness assignment, the population is sorted according to the objective values. The fitness assigned to each individual depends only on the position of the objective values in the individual's rank. Ranking introduces a uniform scaling across the population.

4.3 Tournament selection

In tournament selection, a number of individuals are chosen randomly from the population and the best individual from this group is selected as the parent. This process is repeated as often until there are sufficient individuals to choose. These selected parents produce uniformly random offspring. The tournament size will often depend on the problem, population size, and so on. The parameter for tournament selection is the tournament size. Tournament size takes values ranging from two to the total number of individuals in the population.

4.4 Elitism

When creating a new population by crossover and mutation, there is a big chance that we will lose the best chromosome. Elitism is the name of the method that first copies the best chromosome (or a few best chromosomes) to the new population. The rest is done in the classical way. Elitism can very rapidly increase performance of GA because it prevents losing the best-found solution.

4.5 Genetic operators

Crossover and mutation are two basic operators of GA. Performance of GA depends very much on the genetic operators. Type and implementation of operators depends on encoding and also on the problem. There are many ways of doing crossover and mutation. In this section, we will demonstrate some of the popular methods, with some examples and suggestions as to how to do it for different encoding schemes.

4.6 Crossover

Crossover selects genes from parent chromosomes and creates a new offspring. The simplest way to do this is to choose randomly some crossover point and everything before this point is copied from the first parent and then, everything after a crossover point is copied from the second parent. A single point crossover is illustrated as follows (| is the crossover point):

Chromosome A:	11011|00100110110
Chromosome B:	11011|11000011110
Offspring A:	11011|11000011110
Offspring B:	11011|00100110110

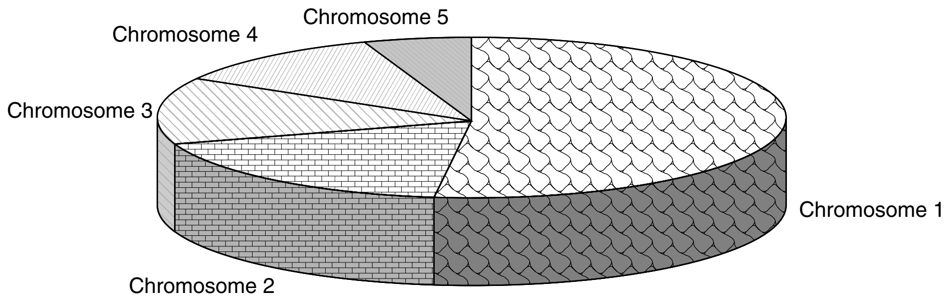

Figure 3. Roulette-wheel selection.

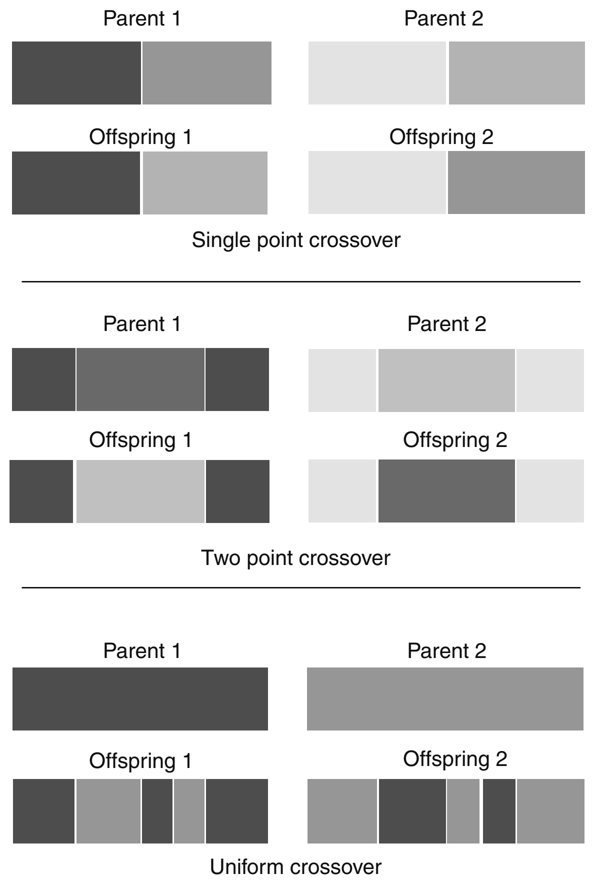

Figure 4. Types of crossover operators.

As illustrated in Figure 4, there are several crossover techniques. In a uniform crossover, bits are randomly copied from the first or the second parent. Specific crossover made for a specific problem can improve the GA performance.

4.7 Mutation

After crossover operation, mutation takes place. Mutation randomly changes the new offspring. For binary encoding, mutation is performed by changing a few randomly chosen bits from 1 to 0 or from 0 to 1. Mutation depends on the encoding as well as the crossover. For example, when we are encoding permutations, mutation could be exchanging two genes. A simple mutation operation is illustrated as follows:

Chromosome A:	1101111000011110
Chromosome B:	1101100100110110
Offspring A:	1100111000011110
Offspring B:	1101101100110110

For many optimization problems, there may be multiple, equal, or unequal optimal solutions. Sometimes, a simple GA cannot maintain stable populations at different optima of such functions. In the case of unequal optimal solutions, the population invariably converges to the global optimum. Niching helps to maintain subpopulations near global and local optima. A niche is viewed as an organism's environment and a species as a collection of organisms with similar features. Niching helps to maintain subpopulations near global and local optima by introducing a controlled competition among different solutions near every local optimal region. Niching is achieved by a sharing function, which creates subdivisions of the environment by degrading an organism's fitness proportional to the number of other members in its neighborhood. The amount of sharing contributed by individuals to their neighbor is determined by their proximity in the decoded parameter space (phenotypic sharing) based on a distance measure (Goldberg, 1989).

5 GA DEMONSTRATIONS

5.1 Rastrigin function

The Rastrigin function is a typical example of nonlinear multimodal function. It was first proposed by Rastrigin (Törn and Zilinskas, 1989) as a two-dimensional function and has later been generalized. This function is a fairly difficult problem due to its large search space and its large number of local minima. Rastrigin's function is defined as

$$F(x) = 10n + \sum_{i=1}^{n} x_i^2 - 10\cos(2\pi x_i), -5.12 \le x_i \le 5.12 \tag{3}$$

The function has just one global minimum, which occurs at the origin where the value of the function is 0. At any local minimum other than [0, 0], the value of Rastrigin's function is greater than 0. The farther the local minimum is from the origin, the larger the value of the function is at that point. Figure 5 illustrates the surface of the function for two input variables.

A real-value representation was used to encode the two input variables. The following parameters were used for the GA experiments.

- Mutation: 0.05, crossover: 0.90
- Population size: 20, number of iterations: 50, selection method: Roulette-wheel selection.

Figure 6 illustrates how the best fitness values were evolved during the 50 generations. As evident after 30 generations, the GA algorithm has succeeded in finding the best optimal solution.

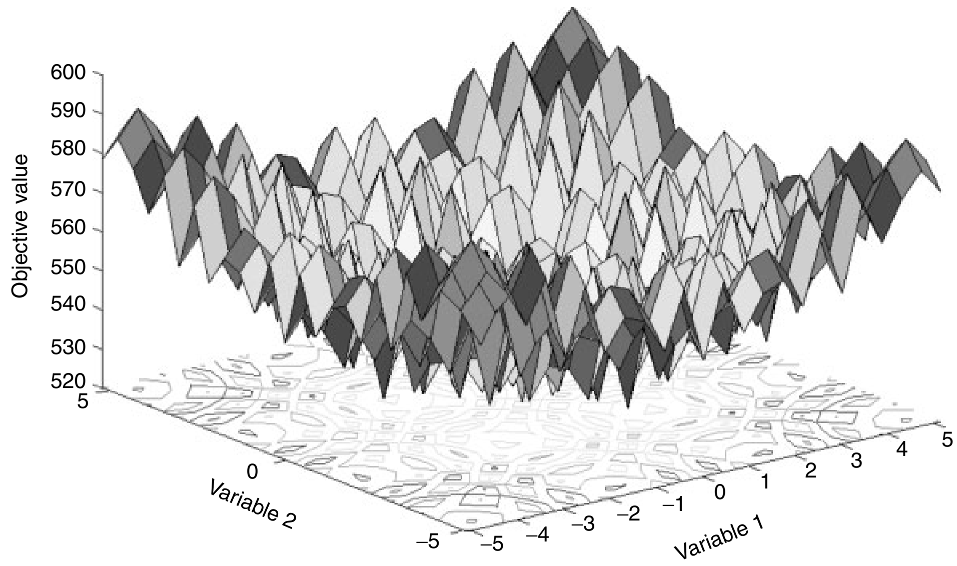

Figure 5. Rastrigin's function for two variables.

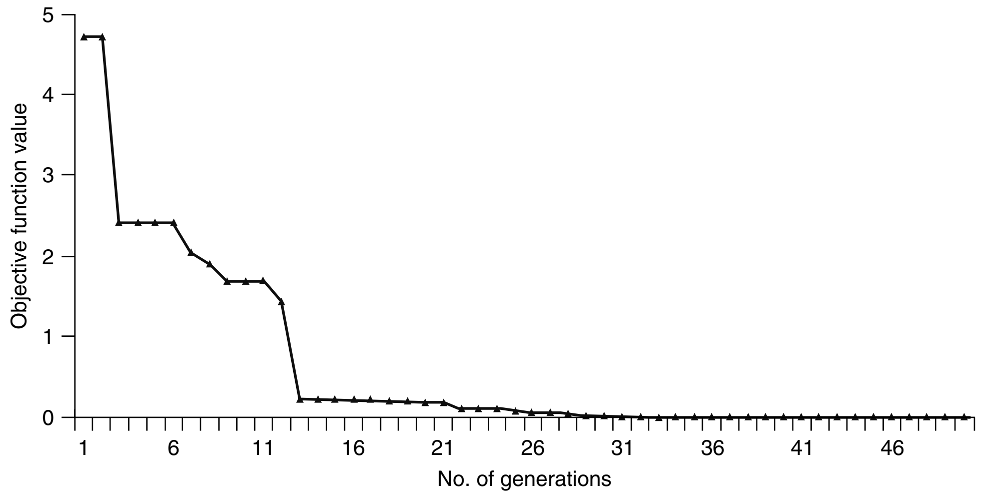

Figure 6. GA learning during the 50 generations.

5.2 Peaks function

Peaks function is a function of two variables, obtained by translating and scaling Gaussian distributions (Jang, Sun and Mizutani, 1997).

$$F(x, y) = 3(1 - x)^2 \exp(-(x^2) - (y + 1)^2) - 10\left(\frac{x}{5 - x^3 - y^5}\right) \exp(-x^2 - y^2) - \tfrac{1}{3}\exp(-(x + 1)^2 - y^2)$$
$$\text{for} - 3 \le x \le 3 \quad \text{and} \quad - 3 \le y \le 3 \tag{4}$$

The Peak function surface is plotted in Figure 7, and the task is to find the optimum value (maximum) for the given range of x and y values. Using a population size of 30, the genetic algorithm was run for 25 iterations. Each input variable was represented using 8 bit. Crossover and mutation rates were set as 0.9 and 0.1 respectively.

Figure 8(a), (b), and (c), illustrate the convergence of the solutions on a contour plot of the surface. After 10 iterations, almost all the solutions were near the optimal point.

6 EVOLUTION STRATEGIES

Evolution strategy (ES) was developed by Rechenberg (1973) and Schwefel (1977). ES tends to be used for

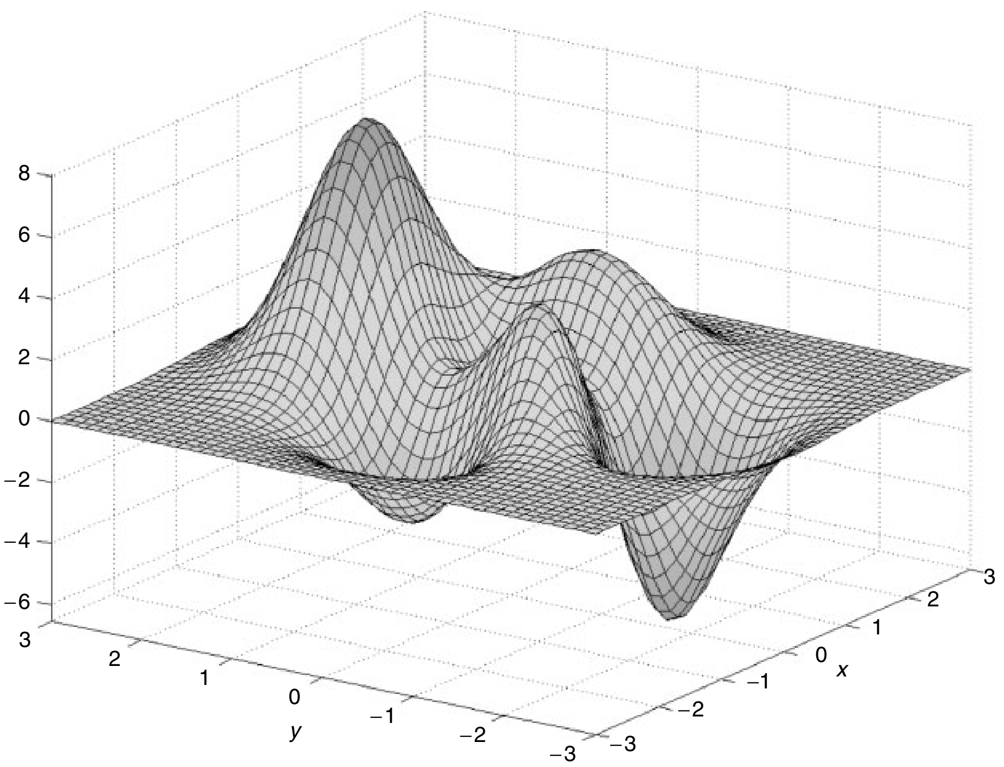

Figure 7. Surface of Peaks function.

empirical experiments that are difficult to model mathematically. The system to be optimized is actually constructed and ES is used to find the optimal parameter settings.

Evolution strategies merely concentrate on translating the fundamental mechanisms of biological evolution for technical optimization problems. The parameters to be optimized are often represented by a vector of real numbers (object parameters – o_p). Another vector of real numbers defines the strategy parameters (s_p), which controls the mutation of the objective parameters. Both object and strategic parameters form the data structure for a single individual.

A population P of n individuals could be described as follows:

$$P = (c_1, c_2, \ldots, c_{n-1}, c_n) \tag{5}$$

where the ith chromosome c_i is defined as: $c_i = (o_p, s_p)$

$$o_p = (o_1, o_2, \ldots, o_{n-1}, o_n) \quad \text{and}$$
$$s_p = (s_1, s_2, \ldots, s_{n-1}, s_n) \tag{6}$$

6.1 Mutation in evolution strategies

The mutation operator is defined as component-wise addition of normal distributed random numbers. Both the objective parameters and the strategy parameters of the chromosome are mutated. Objective parameter vector is calculated as follows:

$$o_{p(mut)} = o_p + N_0(s_p) \tag{7}$$

where $N_0(s_i)$ is the Gaussian distribution of mean value 0 and standard deviation s_i.

Usually, the strategy parameters mutation step size is done by adapting the standard deviation s_i. This may be done (for example) as follows:

$$s_{p(mut)} = (s_1{}^*A_1, s_2{}^*A_2, \ldots, s_{n-1}{}^*A_{n-1}, s_n{}^*A_n) \tag{8}$$

where A_i is randomly chosen from α or $1/\alpha$, depending on the value of equally distributed random variable E of [0,1]

$$A_i = \alpha \quad \text{if} \quad E < 0.5$$
$$A_i = \frac{1}{\alpha} \quad \text{if} \quad E \geq 0.5 \tag{9}$$

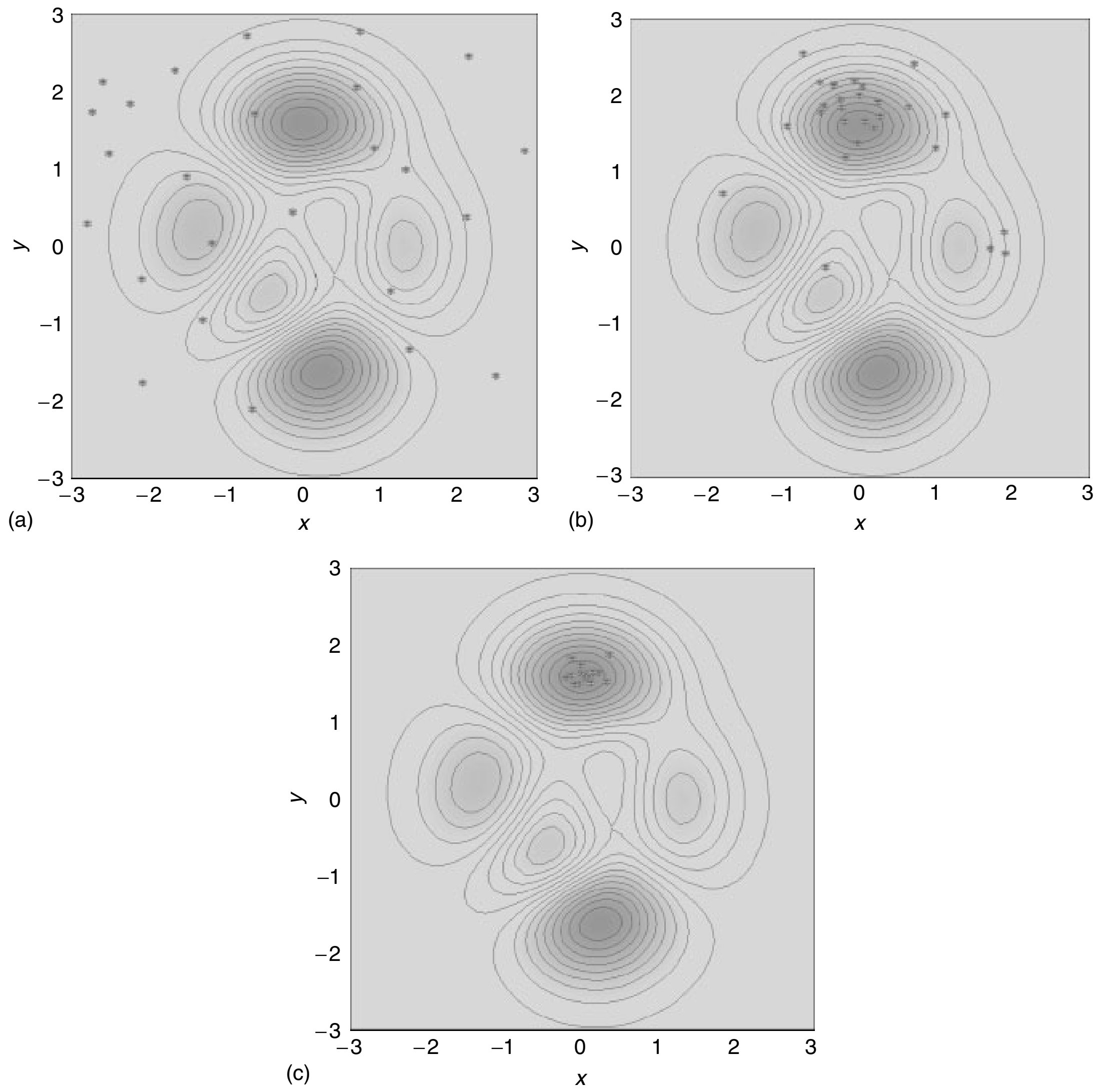

Figure 8. Convergence of solutions (a) generation 0; (b) after 5 generations; (c) after 20 generations (solution points are marked with *).

α is usually referred to as *strategy parameters adaptation value*.

6.2 Crossover (recombination) in evolution strategies

For two chromosomes $c_1 = (o_{p(c1)}, s_{p(c1)})$ and $c_2 = (o_{p(c2)}, s_{p(c2)})$, the crossover operator x is defined as follows:

$$\begin{aligned} R(c_1, c_2) &= c = (o_p, s_p) \\ \text{with } o_{p(i)} &= (o_{p(c1),i} | o_{p(c2),i}) \\ \text{and } s_{p(i)} &= (s_{p(c1),i} | s_{p(c2),i}) \end{aligned} \tag{10}$$

By defining $o_{p(i)}$ and $s_{p(i)} = (x|y)$, a value is randomly assigned for either x or y (50% selection probability for x and y).

6.3 Controlling the evolution

Let P be the number of parents in generation 1 and let C be the number of children in generation i. There are basically four different types of evolution strategies: P,C; $P + C$; $P/R, C$; and $P/R + C$, as discussed below. They mainly differ in how the parents for the next generation are selected and in the usage of crossover operators.

6.3.1 P,C *strategy*

The P parents produce C children, using mutation. Fitness values are calculated for each of the C children and the best P children become next-generation parents. The best individuals of C children are sorted by their fitness value and the first P individuals are selected to be next-generation parents ($C \geq P$).

6.3.2 P + C *strategy*

The P parents produce C children, using mutation. Fitness values are calculated for each of the C children and the best P individuals of both parents and children become next-generation parents. Children and parents are sorted by their fitness value and the first P individuals are selected to be next-generation parents.

6.3.3 P/R, C *strategy*

The P parents produce C children, using mutation and crossover. Fitness values are calculated for each of the C children and the best P children become next-generation parents. The best individuals of C children are sorted by their fitness value and the first P individuals are selected to be next-generation parents ($C \geq P$). Except for the usage of crossover operator, this is exactly the same as P,C strategy.

6.3.4 P/R + C *strategy*

The P parents produce C children, using mutation and recombination. Fitness values are calculated for each of the C children and the best P individuals of both parents and children become next-generation parents. Children and parents are sorted by their fitness value and the first P individuals are selected to be next-generation parents. Except for the usage of crossover operator, this is exactly the same as $P + C$ strategy.

7 EVOLUTIONARY PROGRAMMING

The book *Artificial Intelligence Through Simulated Evolution* by Fogel, Owens and Walsh (1966) is the landmark publication for evolutionary programming (EP). In this book, finite state automata are evolved to predict symbol strings generated from Markov processes and nonstationary time series (AI–FAQ-Genetic). The basic evolutionary programming method involves the following steps:

1. Choose an initial population (possible solutions at random). The number of solutions in a population is highly relevant to the speed of optimization, but no definite answers are available as to how many solutions are appropriate (other than > 1).
2. New offspring are created by mutation. Each offspring solution is assessed by computing its fitness. Typically, a stochastic tournament is held to determine the N solutions to be retained for the population of solutions. It should be noted that, typically, evolutionary programming method does not use any crossover as a genetic operator.

7.1 Evolutionary programming versus genetic algorithms

1. GA is implemented by having arrays of bits or characters to represent the chromosomes. In EP, there are no such restrictions for the representation. In most cases, the representation follows from the problem.
2. EP typically uses an adaptive mutation operator in which the severity of mutations is often reduced as the global optimum is approached, while GAs use a prefixed mutation operator. Among the schemes to adapt the mutation step size, the most widely studied is the 'meta-evolutionary' technique in which the variance of the mutation distribution is subject to mutation by a fixed variance mutation operator that evolves along with the solution.

7.2 Evolutionary programming versus evolution strategies

1. When implemented to solve real-valued function optimization problems, both typically operate on the real values themselves and use adaptive reproduction operators.
2. EP typically uses stochastic tournament selection, while ES typically uses deterministic selection.
3. EP does not use crossover operators, while ES ($P/R, C$ and $P/R + C$ strategies) crossover. However, the effectiveness of the crossover operators depends on the problem at hand.

8 GENETIC PROGRAMMING

The genetic programming (GP) technique provides a framework for automatically creating a working computer program from a high-level statement of the problem (Koza, 1992).

Genetic programming achieves this goal of automatic programming by genetically breeding a population of computer programs, using the principles of Darwinian natural selection and biologically inspired operations. The operations include most of the techniques discussed in the previous sections. The main difference between GP and GA is the representation of the solution. GP creates computer programs in the LISP or scheme computer languages as the solution. LISP is an acronym for LISt Processor and was developed in the late 1950s (History of LISP, 2004). Unlike most languages, LISP is usually used as an interpreted language. This means that, unlike compiled languages, an interpreter can process and respond directly to programs written in LISP.

The main reason for choosing LISP to implement GP is because of the advantage that the programs and data have the same structure, which could provide easy means for manipulation and evaluation.

In GP, the individual population members are not fixed-length character strings that encode possible solutions to the problem at hand, they are programs that, when executed, are the candidate solutions to the problem. These programs are expressed in genetic programming as parse trees rather than as lines of code. For example, the simple program '$a + b * c$' would be represented as shown in Figure 9. The terminal and function sets are also important components of genetic programming. The terminal and function sets are the alphabets of the programs to be made. The terminal set consists of the variables and constants of the programs (e.g. A, B, and C in Figure 9).

The most common way of writing down a function with two arguments is the infix notation. That is, the two arguments are connected with the operator symbol between them as follows.

$$A + B$$

A different method is the prefix notation. Here, the operator symbol is written down first, followed by its required arguments.

$$+AB$$

While this may be a bit more difficult or just unusual for human eyes, it opens some advantages for computational uses. The computer language LISP uses symbolic expressions (or S-expressions) composed in prefix notation. Then, a simple S-expression could be

$$(\mathit{operator, argument}),$$

where *operator* is the name of a function and *argument* can be either a constant or a variable or another symbolic expression, as shown below.

$$(\mathit{operator, argument}(\mathit{operator, argument})$$
$$(\mathit{operator, argument}))$$

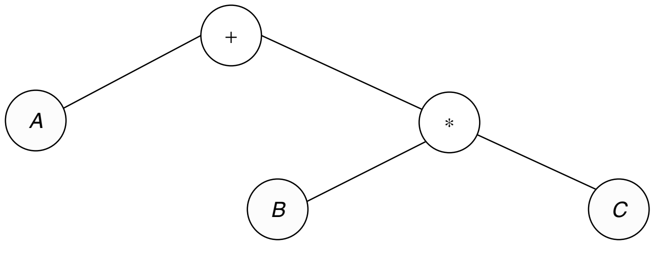

Figure 9. A simple tree structure of GP.

9 GENETIC PROGRAMMING BASICS

A parse tree is a structure that develops the interpretation of a computer program. Functions are written down as nodes and their arguments as leaves. A subtree is the part of a tree that is under an inner node of this tree, as illustrated in Figure 10. If this tree is cut out from its parent, the inner node becomes a root node and the subtree is a valid tree of its own.

There is a close relationship between these parse trees and S-expression; in fact, these trees are just another way of writing down expressions. While functions will be the nodes of the trees (or the operators in the S-expressions) and can have other functions as their arguments, the leaves will be formed by terminals, that is, symbols that may not be further expanded. Terminals can be variables, constants, or specific actions that are to be performed. The process of selecting the functions and terminals that are needed or are useful for finding a solution to a given problem is one of the key steps in GP. Evaluation of these structures is straightforward. Beginning at the root node, the values of all subexpressions (or subtrees) are computed, descending the tree down to the leaves. GP procedure could be summarized as follows:

- generate an initial population of random compositions of the functions and terminals of the problem;
- compute the fitness values of each individual in the population;
- using some selection strategy and suitable reproduction operators, produce offsprings;
- iterate the procedure until the required solution is found or the termination conditions have been reached (specified number of generations).

The creation of an offspring from the crossover operation is accomplished by deleting the crossover fragment of the first parent and then inserting the crossover fragment of the second parent. The second offspring is produced in a symmetric manner. A simple crossover operation is illustrated in Figure 11. In GP, the crossover operation is

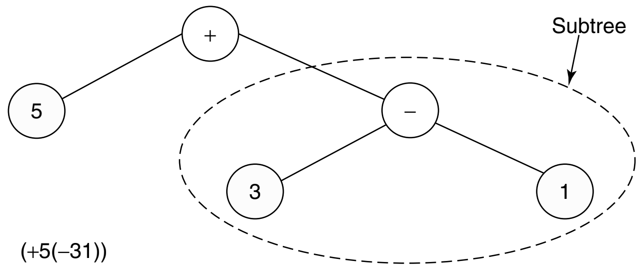

Figure 10. Illustration of a parse tree and a subtree.

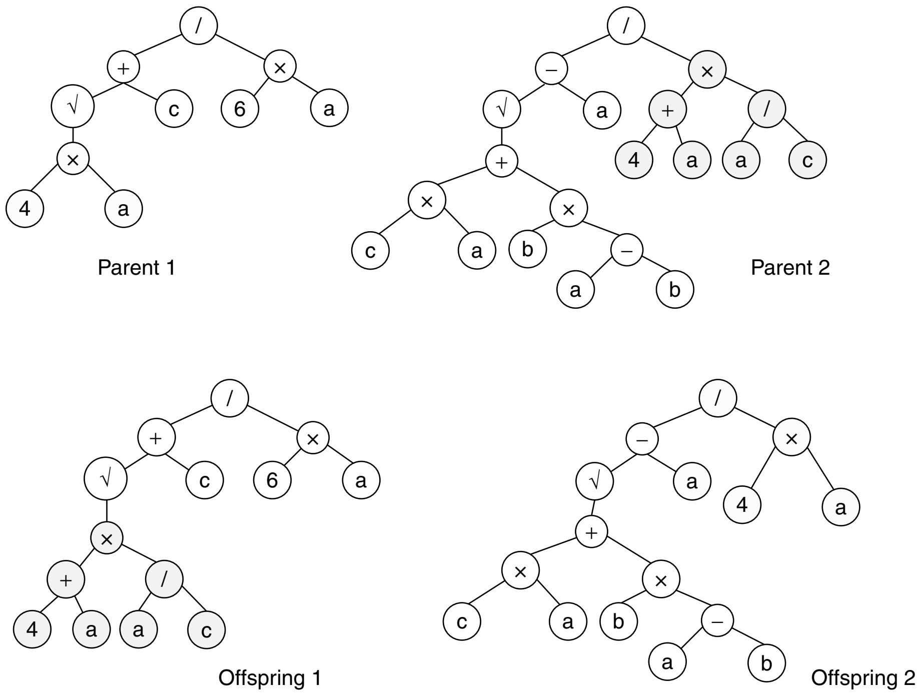

Figure 11. Illustration of crossover operator.

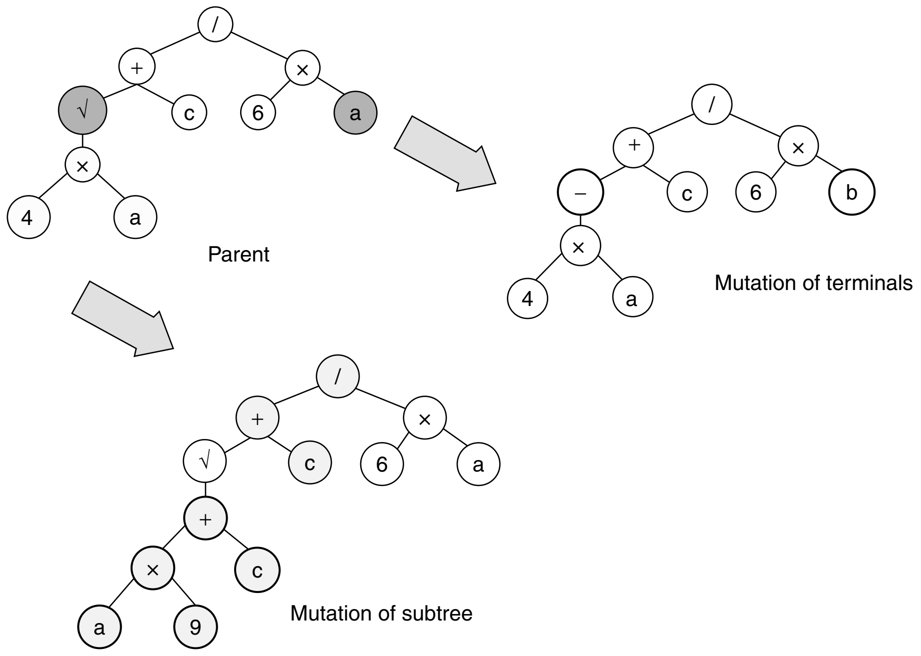

Figure 12. Illustration of mutation operator in GP.

implemented by taking randomly selected subtrees in the individuals and exchanging them.

Mutation is another important feature of genetic programming. Two types of mutations are commonly used. The simplest type is to replace a function or a terminal by another function or a terminal respectively. In the second kind, an entire subtree can replace another subtree. Figure 12 explains the concepts of mutation.

GP requires data structures that are easy to handle and evaluate and are robust to structural manipulations. These are among the reasons why the class of S-expressions was chosen to implement GP. The set of functions and terminals that will be used in a specific problem has to be chosen carefully. If the set of functions is not powerful enough, a solution may be very complex or may not be found at all. Like in any evolutionary computation technique, the generation of the first population of individuals is important for successful implementation of GP. Some of the other factors that influence the performance of the algorithm are the size of the population, percentage of individuals that participate in the crossover/mutation, maximum depth for the initial individuals and the maximum allowed depth for the generated offspring, and so on. Some specific advantages of genetic programming are that no analytical knowledge is needed and still accurate results could be obtained. GP approach does scale with the problem size. GP does impose restrictions on how the structure of solutions should be formulated.

10 SUMMARY

This article presents the biological motivation and fundamental aspects of evolutionary algorithms and its constituents, namely, genetic algorithm, evolution strategies, evolutionary programming, and genetic programming. Performance of genetic algorithms is demonstrated using two function optimization problems. Important advantages of evolutionary computation as compared to classical optimization techniques are also discussed.

REFERENCES

AI – FAQ-Genetic. http://www.faqs.org/faqs/ai-faq/genetic/, accessed on September 10, 2004.

Bäck, T. (1996) *Evolutionary Algorithms in Theory and Practice: Evolution Strategies, Evolutionary Programming, Genetic algorithms*, Oxford University Press, New York.

Fogel, D.B. (1999) *Evolutionary Computation: Toward a New Philosophy of Machine Intelligence*, 2nd edn, IEEE Press, Piscataway, NJ.

Fogel, L.J., Owens, A.J. and Walsh, M.J. (1966) *Artificial Intelligence Through Simulated Evolution*, John Wiley & Sons, New York.

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

History of LISP. (2004) http://www-formal.stanford.edu/jmc/history/lisp/lisp.html.

Holland, J. (1975) *Adaptation in Natural and Artificial Systems*, University of Michican Press, Ann Harbor, MI.

Jang, J.S.R., Sun, C.T. and Mizutani, E. (1997) *Neuro-Fuzzy and Soft Computing: A Computational Approach to Learning and Machine Intelligence*, Prentice Hall, USA.

Koza, J.R. (1992) *Genetic Programming*, MIT Press, Cambridge, MA.

Rechenberg, I. (1973) *Evolutionsstrategie: Optimierung Technischer Systeme nach Prinzipien der biologischen Evolution*, Fromman-Holzboog, Stuttgart.

Schwefel, H.P. (1977) *Numerische Optimierung von Computermodellen Mittels der Evolutionsstrategie*, Birkhaeuser, Basel.

Törn, A. and Zilinskas, A. (1989) *Global Optimization, Lecture Notes in Computer Science*, Vol. 350, Springer-Verlag, Berlin.

FURTHER READING

Michalewicz, Z. (1992) *Genetic Algorithms + Data Structures = Evolution Programs*, Springer-Verlag, Berlin.