15 Genetic Algorithm

Learning Objectives

- Gives an introduction to natural evolution.
- Lists the basic operators (selection, crossover, mutation) and other terminologies used in Genetic Algorithms (GAs).
- Discusses the need for schemata approach.
- Details the comparison of traditional algorithm with GA.
- Explains the operational flow of simple GA.
- Description is given of the various classifications of GA – Messy GA, adaptive GA, hybrid GA, parallel GA and independent sampling GA.
- The variants of parallel GA (fine-grained parallel GA and coarse-grained parallel GA) are included.
- Enhances the basic concepts involved in Holland classifier system.
- The various features and operational properties of genetic programming are provided.
- The application areas of GA are also discussed.

Charles R. Darwin says that "Although the belief that an organ so perfect as the eye could have been formed by natural selection is enough to stagger any one; yet in the case of any organ, if we know of a long series of gradations in complexity, each good for its possessor, then, under changing conditions of life, there is no logical impossibility in the acquirement of any conceivable degree of perfection through natural selection."

15.1 Introduction

Charles Darwin has formulated the fundamental principle of natural selection as the main evolutionary tool. He put forward his ideas without the knowledge of basic hereditary principles. In 1865, Gregor Mendel discovered these hereditary principles by the experiments he carried out on peas. After Mendel's work genetics was developed. Morgan experimentally found that chromosomes were the carriers of hereditary information and that genes representing the hereditary factors were lined up on chromosomes. Darwin's natural selection theory and natural genetics remained unlinked until 1920s when it was proved that genetics and selection were in no way contrasting each other. Combination of Darwin's and Mendel's ideas lead to the modern evolutionary theory.

In *The Origin of Species*, Charles Darwin stated the theory of natural evolution. Over many generations, biological organisms evolve according to the principles of natural selection like "survival of the fittest" to reach some remarkable forms of accomplishment. The perfect shape of the albatross wing, the efficiency and the similarity between sharks and dolphins and so on are good examples of what random evolution with absence of intelligence can achieve. So, if it works so well in nature, it should be interesting to simulate natural evolution and try to obtain a method which may solve concrete search and optimization problems.

For a better understanding of this theory, it is important first to understand the biological terminology used in evolutionary computation. It is discussed in Section 15.2.

In 1975, Holland developed this idea in *Adaptation in Natural and Artificial Systems*. By describing how to apply the principles of natural evolution to optimization problems, he laid down the first GA. Holland's theory has been further developed and now GAs stand up as powerful adaptive methods to solve search and optimization problems. Today, GAs are used to resolve complicated optimization problems, such as, organizing the time table, scheduling job shop, playing games.

15.1.1 What are Genetic Algorithms?

GAs are adaptive heuristic search algorithms based on the evolutionary ideas of natural selection and genetics. As such they represent an intelligent exploitation of a random search used to solve optimization problems. Although randomized, GAs are by no means random; instead they exploit historical information to direct the search into the region of better performance within the search space. The basic techniques of the GAs are designed to simulate processes in natural systems necessary for evolution, especially those that follow the principles first laid down by Charles Darwin, "survival of the fittest," because in nature, competition among individuals for scanty resources results in the fittest individuals dominating over the weaker ones.

15.1.2 Why Genetic Algorithms?

They are better than conventional algorithms in that they are more robust. Unlike older AI systems, they do not break easily even if the inputs are changed slightly or in the presence of reasonable noise. Also, in searching a large state-space, multimodal state-space or n-dimensional surface, a GA may offer significant benefits over more typical optimization techniques (linear programming, heuristic, depth-first, breath-first and praxis.)

15.2 Biological Background

The science that deals with the mechanisms responsible for similarities and differences in a species is called Genetics. The word "genetics" is derived from the Greek word "genesis" meaning "to grow" or "to become." The science of genetics helps us to differentiate between heredity and variations and accounts for the resemblances and differences during the process of evolution. The concepts of GAs are directly derived from natural evolution and heredity. The terminologies involved in the biological background of species are discussed in the following subsections.

15.2.1 The Cell

Every animal/human cell is a complex of many "small" factories that work together. The center of all this is the cell nucleus. The genetic information is contained in the cell nucleus. Figure 15-1 shows anatomy of the animal cell and cell nucleus.

15.2.2 Chromosomes

All the genetic information gets stored in the chromosomes. Each chromosome is build of deoxyribonucleic acid (DNA). In humans, chromosomes exist in pairs (23 pairs found). The chromosomes are divided into several parts called *genes*. Genes code the properties of species, i.e., the characteristics of an individual. The possibilities of combination of the genes for one property are called *alleles*, and a gene can take different alleles. For example, there is a gene for eye color, and all the different possible alleles are black, brown, blue and green (since no one has red or violet eyes!). The set of all possible alleles present in a particular population forms a *gene pool*. This gene pool can determine all the different possible variations for the future generations. The size of the gene pool helps in determining the diversity of the individuals in the population. The set of all the genes of a specific species is called *genome*. Each and every gene has a unique position on the genome called

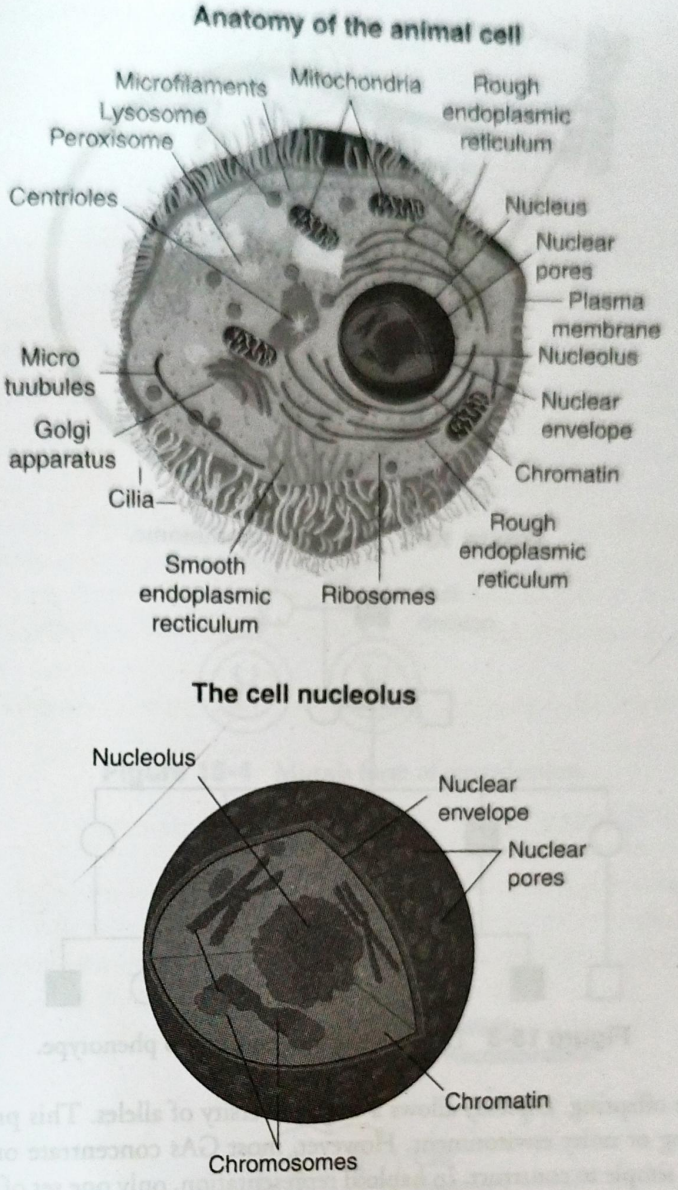


Figure 15-1 Anatomy of animal cell, cell nucleus.

locus. In fact, most living organisms store their genome on several chromosomes, but in the GAs, all the genes are usually stored on the same chromosomes. Thus, chromosomes and genomes are synonyms with one other in GAs. Figure 15-2 shows a model of chromosome.

15.2.3 Genetics

For a particular individual, the entire combination of genes is called *genotype*. The *phenotype* describes the physical aspect of decoding a genotype to produce the phenotype. One interesting point of evolution is that selection is always done on the phenotype whereas the reproduction recombines genotype. Thus, morphogenesis plays a key role between selection and reproduction. In higher life forms, chromosomes contain two sets of genes. These are known as *diploids*. In the case of conflicts between two values of the same pair of genes, the dominant one will determine the phenotype whereas the other one, called *recessive*, will still be present and

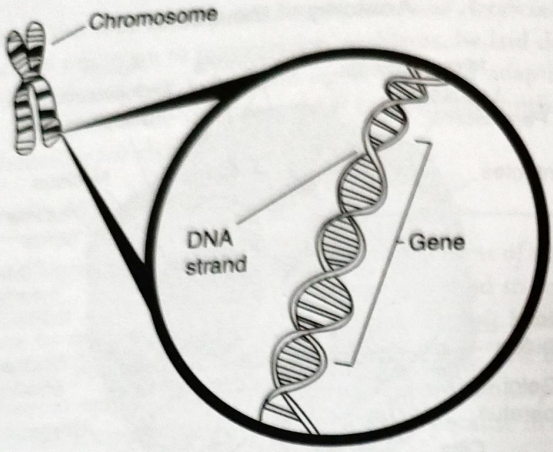


Figure 15-2 Model of chromosome.

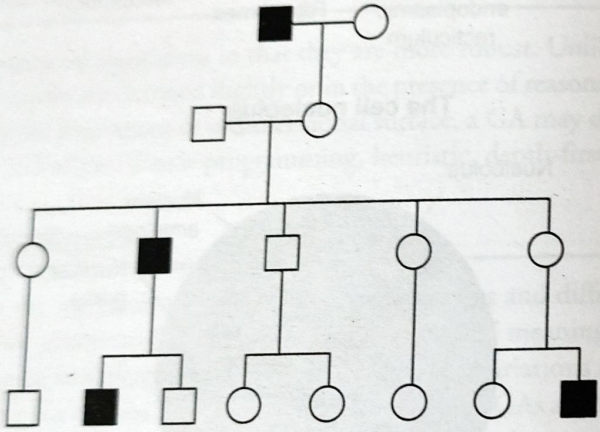


Figure 15-3 Development of genotype to phenotype.

can be passed onto the offspring. *Diploidy* allows a wider diversity of alleles. This provides a useful memory mechanism in changing or noisy environment. However, most GAs concentrate on haploid chromosomes because they are much simple to construct. In haploid representation, only one set of each gene is stored, thus the process of determining which allele should be dominant and which one should be recessive is avoided. Figure 15-3 shows the development of genotype to phenotype.

15.2.4 Reproduction

Reproduction of species via genetic information is carried out by the following:

1. *Mitosis*: In mitosis the same genetic information is copied to new offspring. There is no exchange of information. This is a normal way of growing of multicell structures, such as organs. Figure 15-4 shows mitosis form of reproduction.
2. *Meiosis*: Meiosis forms the basis of sexual reproduction. When meiotic division takes place, two gametes appear in the process. When reproduction occurs, these two gametes conjugate to a zygote which becomes the new individual. Thus in this case, the genetic information is shared between the parents in order to create new offspring. Figure 15-5 shows meiosis form of reproduction.

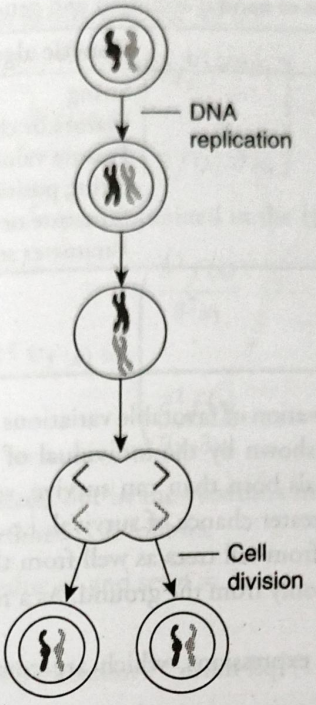


Figure 15-4 Mitosis form of reproduction.

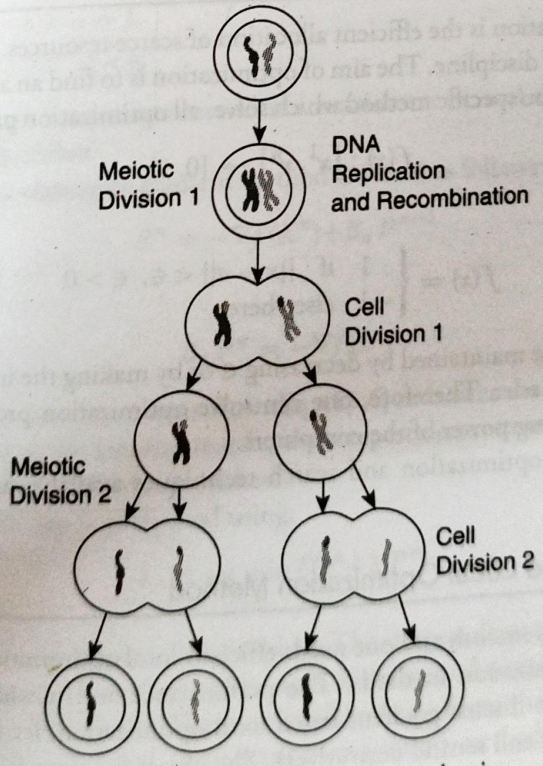


Figure 15-5 Meiosis form of reproduction.

Table 15-1 Comparison of natural evolution and genetic algorithm terminology

Natural evolution	Genetic algorithm
Chromosome	String
Gene	Feature or character
Allele	Feature value
Locus	String position
Genotype	Structure or coded string
Phenotype	Parameter set, a decoded structure

15.2.5 Natural Selection

The origin of species is based on "Preservation of favorable variations and rejection of unfavorable variations." The variation refers to the differences shown by the individual of a species and also by offspring's of the same parents. There are more individuals born than can survive, so there is a continuous struggle for life. Individuals with an advantage have a greater chance of survival, i.e., the survival of the fittest. For example, Giraffe with long necks can have food from tall trees as well from the ground; on the other hand, goat and deer having smaller neck can have food only from the ground. As a result, natural selection plays a major role in this survival process.

Table 15.1 gives a list of different expressions, which are common in natural evolution and genetic algorithm.

15.3 Traditional Optimization and Search Techniques

The basic principle of optimization is the efficient allocation of scarce resources. Optimization can be applied to any scientific or engineering discipline. The aim of optimization is to find an algorithm which solves a given class of problems. There exists no specific method which solves all optimization problems. Consider a function,

$$f(x) : [x^1, x^u] \rightarrow [0, 1] \tag{15.1}$$

where

$$f(x) = \begin{cases} 1 & \text{if } ||x - a|| < \in,\ \in > 0 \\ -1 & \text{elsewhere} \end{cases}$$

For the above function, f can be maintained by decreasing $\in$ or by making the interval of $[x^1, x^u]$ large. Thus, a difficult task can be made easier. Therefore, one can solve optimization problems by combining human creativity and the raw processing power of the computers.

The various conventional optimization and search techniques available are discussed in the following subsections.

15.3.1 Gradient-Based Local Optimization Method

When the objective function is smooth and one needs efficient local optimization, it is better to use gradient-based or Hessian-based optimization methods. The performance and reliability of the different gradient methods vary considerably. To discuss gradient-based local optimization, let us assume a smooth objective function (i.e., continuous first and second derivatives). The object function is denoted by

$$f(x) : R^n \rightarrow R \tag{15.2}$$

The first derivatives are contained in the gradient vector $\nabla f(x)$

$$\nabla f(x) = \begin{bmatrix} \partial f(x)/\partial x_1 \\ \vdots \\ \partial f(x)/\partial x_n \end{bmatrix} \tag{15.3}$$

The second derivatives of the object function are contained in the Hessian matrix $H(x)$:

$$H(x) = \nabla^{\mathrm{T}} \nabla f(x) = \begin{pmatrix} \dfrac{\partial^2 f(x)}{\partial^2 x_1} & \cdots & \dfrac{\partial^2 f(x)}{\partial x_1 \partial x_n} \\ \vdots & & \vdots \\ \dfrac{\partial^2 f(x)}{\partial x_1 \partial x_n} & \cdots & \dfrac{\partial^2 f(x)}{\partial^2 x_n} \end{pmatrix} \tag{15.4}$$

Few methods need only the gradient vector, but in the Newton's method we need the Hessian matrix. The general pseudocode used in gradient methods is as follows:

Select an initial guess value x^1 and set $n = 1$.

Repeat

Solve the search direction P^n from Eq. (15.5) or (15.6) below.

Determine the next iteration point using Eq. (15.7) below:

$$X^{n+1} = X^n + \lambda_n P^n$$

Set $n = n + 1$.

Until $\|X^n - X^{n-1}\| < \epsilon$

These gradient methods search for minimum and not maximum. Several different methods are obtained based on the details of the algorithm.

The search direction P^n in conjugate gradient method is found as follows:

$$P^n = -\nabla f(X^n) + \beta_n P^{n-1} \tag{15.5}$$

In secant method,

$$B_n P^n = -\nabla f(x^n) \tag{15.6}$$

is used for finding search direction. The matrix B_n in Eq. (15.6) estimates the Hessian and is updated in each iteration. When B_n is defined as the identity matrix, the steepest descent method occurs. When the matrix B_n is the Hessian $H(x^n)$, we get the Newton's method.

The length λ_n of the search step is computed using:

$$\lambda_n = \arg\min_{\lambda > 0} f(x^n + \lambda P^n) \tag{15.7}$$

The discussed is a one-dimensional optimization problem. The steepest descent method provides poor performance. As a result, conjugate gradient method can be used. If the second derivatives are easy to compute, then Newton's method may provide best results. The secant methods are faster than conjugate gradient methods, but there occurs memory problems. Thus, these local optimization methods can be combined with other methods to get a good link between performance and reliability.

15.3.2 Random Search

Random search is an extremely basic method. It only explores the search space by randomly selecting solutions and evaluates their fitness. This is quite an unintelligent strategy, and is rarely used. Nevertheless, this method is sometimes worth testing. It doesn't take much effort to implement it, and an important number of evaluations can be done fairly quickly. For new unresolved problems, it can be useful to compare the results of a more advanced algorithm to those obtained just with a random search for the same number of evaluations. Nasty surprises might well appear when comparing, for example, GAs to random search. It's good to remember that the efficiency of GA is extremely dependent on consistent coding and relevant reproduction operators. Building a GA which performs no more than a random search happens more often than we can expect. If the reproduction operators are just producing new random solutions without any concrete links to the ones selected from the last generation, the GA is just doing nothing else than a random search.

Random search does have a few interesting qualities. However good the obtained solution may be, if it's not optimal one, it can be always improved by continuing the run of the random search algorithm for long enough. A random search never gets stuck at any point such as a local optimum. Furthermore, theoretically, if the search space is finite, random search is guaranteed to reach the optimal solution. Unfortunately, this result is completely useless. For most of problems we are interested in, exploring the whole search space takes a lot of time.

15.3.3 Stochastic Hill Climbing

Efficient methods exist for problems with well-behaved continuous fitness functions. These methods use a kind of gradient to guide the direction of search. *Stochastic hill climbing* is the simplest method of these kinds. Each iteration consists in choosing randomly a solution in the neighborhood of the current solution and retains this new solution only if it improves the fitness function. Stochastic hill climbing converges towards the optimal solution if the fitness function of the problem is continuous and has only one peak (unimodal function).

On functions with many peaks (multimodal functions), the algorithm is likely to stop on the first peak it finds even if it is not the highest one. Once a peak is reached, hill climbing cannot progress anymore, and that is problematic when this point is a local optimum. Stochastic hill climbing usually starts from a random select point. A simple idea to avoid getting stuck on the first local optimal consists in repeating several hill climbs each time starting from a different randomly chosen point. This method is sometimes known as *iterated hill climbing*. By discovering different local optimal points, chances to reach the global optimum increase. It works well if there are not too many local optima in the search space. However, if the fitness function is very "noisy" with many small peaks, stochastic hill climbing is definitely not a good method to use. Nevertheless, such methods have the advantage of being easy to implement and giving fairly good solutions very quickly.

15.3.4 Simulated Annealing

Simulated annealing (SA) was originally inspired by formation of crystal in solids during cooling. As discovered a long time ago by Iron Age blacksmiths, the slower the cooling, the more perfect is the crystal formed. By cooling, complex physical systems naturally converge towards a state of minimal energy. The system moves randomly, but the probability to stay in a particular configuration depends directly on the energy of the system

and on its temperature. This probability is formally given by Gibbs law:

$$p = e^{E/kT} \tag{15.8}$$

where E stands for the energy, k is the Bolttzmann constant and T is the temperature. In the mid01970s, Kirkpatrick by analogy of this physical phenomena, laid out the first description of SA.

As in the stochastic hill climbing, the iteration of the SA consists of randomly choosing a new solution in the neighborhood of the actual solution. If the fitness function of the new solution is better than the fitness function of the current one, the new solution is accepted as the new current solution. If the fitness function is not improved, the new solution is retained with a probability:

$$p = e^{-[f(y)-f(x)]/kT} \tag{15.9}$$

where $f(y) - f(x)$ is the difference of the fitness function between the new and the old solution.

The SA behaves like a hill climbing method but with the possibility of going downhill to avoid being trapped at local optima. When the temperature is high, the probability of deteriorate the solution is quite important, and then a lot of large moves are possible to explore the search space. The more the temperature decreases, the more difficult it is to go downhill. The algorithm thus tries to climb up from the current solution to reach a maximum. When temperature is lower, there is an exploitation of the current solution. If the temperature is too low, number deterioration is accepted, and the algorithm behaves just like a stochastic hill climbing method. Usually, the SA starts from a high temperature which decreases exponentially. The slower the cooling, the better it is for finding good solutions. It even has been demonstrated that with an infinitely slow cooling, the algorithm is almost certain to find the global optimum. The only point is that infinitely slow cooling consists in finding the appropriate temperature decrease rate to obtain a good behavior of the algorithm.

SA by mixing exploration features such as the random search and exploitation features like hill climbing usually gives quite good results. SA is a serious competitor of GAs. It is worth trying to compare the results obtained by each. Both are derived from analogy with natural system evolution and both deal with the same kind of optimization problem. GAs differ from SA in two main features which makes them more efficient. First, GAs use a population-based selection whereas SA only deals with one individual at each iteration. Hence GAs are expected to cover a much larger landscape of the search space at each iteration; however, SA iterations are much more simple, and so, often much faster. The great advantage of GA is its exceptional ability to be parallelized, whereas SA does not gain much of this. It is mainly due to the population scheme use by GA. Second, GAs use recombination operators, and are able to mix good characteristics from different solutions. The exploitation made by recombination operators are supposedly considered helpful to find optimal solutions of the problem. On the other hand, SA is still very simple to implement and gives good results. SAs have proved their efficiency over a large spectrum of difficult problems, like the optimal layout of printed circuit board or the famous traveling salesman problem.

15.3.5 Symbolic Artificial Intelligence

Most symbolic artificial intelligence (AI) systems are very static. Most of them can usually only solve one given specific problem, since their architecture was designed for whatever that specific problem was in the first place. Thus, if the given problem were somehow to be changed, these systems could have a hard time adapting to them, since the algorithm that would originally arrive to the solution may be either incorrect or less efficient. GAs were created to combat these problems. They are basically algorithms based on natural biological evolution. The architecture of systems that implement GAs is more able to adapt to a wide range of

problems. A GA functions by generating a large set of possible solutions to a given problem. It then evaluates each of those solutions, and decides on a "fitness level" (you may recall the phrase: "survival of the fittest") for each solution set. These solutions then breed new solutions. The parent solutions that were more "fit" are more likely to reproduce, while those that were less "fit" are more unlikely to do so. In essence, solutions are evolved over time. This way we evolve our search space scope to a point where you can find the solution. GAs can be incredibly efficient if programmed correctly.

15.4 Genetic Algorithm and Search Space

Evolutionary computing was introduced in the 1960s by I. Rechenberg in the work "Evolution Strategies." This idea was then developed by other researches. GAs were invented by John Holland and developed this idea in his book "Adaptation in Natural and Artificial Systems" in the year 1975. Holland proposed GA as a heuristic method based on "survival of the fittest." GA was discovered as a useful tool for search and optimization problems.

15.4.1 Search Space

Most often one is looking for the best solution in a specific set of solutions. The space of all feasible solutions (the set of solutions among which the desired solution resides) is called *search space* (also state space). Each and every point in the search space represents one possible solution. Therefore, each possible solution can be "marked" by its fitness value, depending on the problem definition. With GA one looks for the best solution among a number of possible solutions – represented by one point in the search space; GAs are used to search the search space for the best solution, e.g., minimum. The difficulties in this case are the local minima and the starting point of the search. Figure 15-6 gives an example of search space.

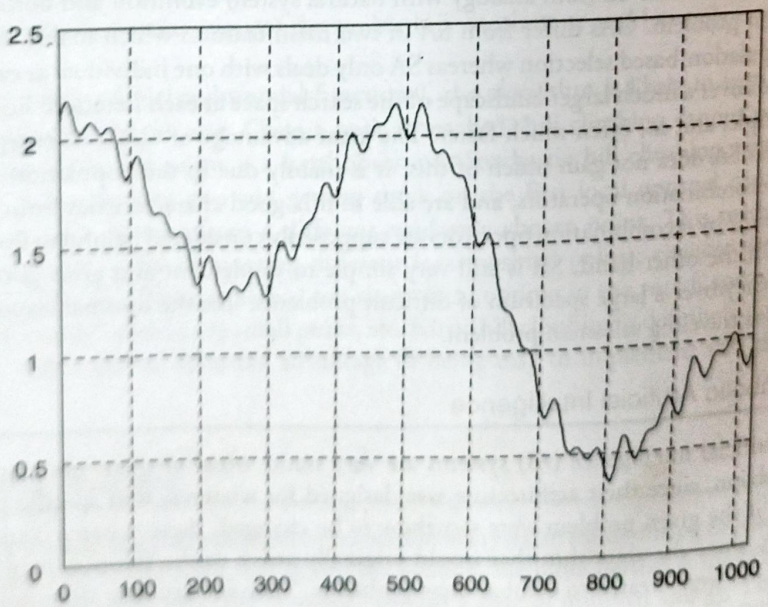


Figure 15-6 An example of search space.

15.4.2 Genetic Algorithms World

GA raises again a couple of important features. First, it is a *stochastic* algorithm; randomness has an essential role in GAs. Both selection and reproduction need random procedures. A second very important point is that GAs always consider a population of solutions. Keeping in memory more than a single solution at each iteration offers a lot of advantages. The algorithm can recombine different solutions to get better ones and so it can use the benefits of assortment. A population-based algorithm is also very amenable for parallelization. The *robustness* of the algorithm should also be mentioned as something essential for the algorithm's success. Robustness refers to the ability to perform consistently well on a broad range of problem types. There is no particular requirement on the problem before using GAs, so it can be applied to resolve any problem. All these features make GA a really powerful optimization tool.

With the success of GAs, other algorithms making use of the same principle of natural evolution have also emerged. Evolution strategy, genetic programming are some algorithms similar to these algorithms. The classification is not always clear between the different algorithms, thus to avoid any confusion, they are all gathered in what is called *Evolutionary Algorithms*.

The analogy with nature gives these algorithms something exciting and enjoyable. Their ability to deal successfully with a wide range of problem area, including those which are difficult for other methods to solve makes them quite powerful. However today, GAs are suffering from too much trendiness. GA is a new field, and parts of the theory still have to be properly established. We can find almost as many opinions on GAs as there are researchers in this field. In this document, we will generally find the most current point of view. But things evolve quickly in GAs too, and some comments might not be very accurate in few years.

It is also important to mention GA limits in this introduction. Like most stochastic methods, GAs are not guaranteed to find the global optimum solution to a problem; they are satisfied with finding "acceptably good" solutions to the problem. GAs are extremely general too, and so specific techniques for solving particular problems are likely to out-perform GAs in both speed and accuracy of the final result. GAs are something worth trying when everything else fails or when we know absolutely nothing of the search space. Nevertheless, even when such specialized techniques exist, it is often interesting to hybridize them with a GA in order to possibly gain some improvements. It is important always to keep an objective point of view; do not consider that GAs are a panacea for resolving all optimization problems. This warning is for those who might have the temptation to resolve anything with GA. The proverb says "If we have a hammer, all the problems look like a nails." GAs do work and give excellent results if they are applied properly on appropriate problems.

15.4.3 Evolution and Optimization

To depict the importance of evolution and optimization process, consider a species Basilosaurus that originated 45 million years ago. The Basilosaurus was a prototype of a whale (Figure 15-7). It was about 15 m long and

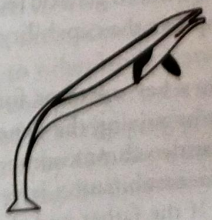


Figure 15-7 Basilosaurus.

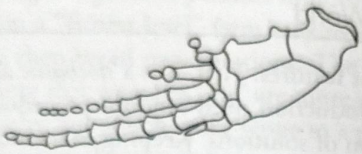


Figure 15-8 Tursiops flipper.

weighed approximately 5 tons. It still had a quasi-independent head and posterior paws, and moved using undulatory movements and hunted small preys. Its anterior members were reduced to small flippers with an elbow articulation. Movements in such a viscous element (water) are very hard and require big efforts. The anterior members of basilosaurus were not really adapted to swimming. To adapt them, a double phenomenon must occur: the shortening of the "arm" with the locking of the elbow articulation and the extension of the fingers constitute the base structure of the flipper (refer Figure 15-8).

The image shows that two fingers of the common dolphin are hypertrophied to the detriment of the rest of the member. The basilosaurus was a hunter; it had to be fast and precise. Through time, subjects appeared with longer fingers and short arms. They could move faster and more precisely than before, and therefore, live longer and have many descendants.

Meanwhile, other improvements occurred concerning the general aerodynamic like the integration of the head to the body, improvement of the profile, strengthening of the caudal fin, and so on, finally producing a subject perfectly adapted to the constraints of an aqueous environment. This process of adaptation and this morphological optimization is so perfect that nowadays the similarity between a shark, a dolphin or a submarine is striking. The first is a cartilaginous fish (Chondrichtyen) that originated in the Devonian period (~400 million years), long before the apparition of the first mammal. Darwinian mechanism hence generated an optimization process – *hydrodynamic optimization* – for fishes and others marine animals – *aerodynamic optimization* for pterodactyls, birds and bats. This observation is the basis of GAs.

15.4.4 Evolution and Genetic Algorithms

The basic idea is as follows: the genetic pool of a given population potentially contains the solution, or a better solution, to a given adaptive problem. This solution is not "active" because the genetic combination on which it relies is split among several subjects. Only the association of different genomes can lead to the solution. Simplistically speaking, we could by example consider that the shortening of the paw and the extension of the fingers of our basilosaurus are controlled by two "genes." No subject has such a genome, but during reproduction and crossover, new genetic combination occur and, finally, a subject can inherit a "good gene" from both parents his paw is now a flipper.

Holland method is especially effective because he not only considered the role of mutation (mutations improve very seldom the algorithms), but also utilized genetic recombination (crossover): these recombination, the crossover of partial solutions, greatly improve the capability of the algorithm to approach, and eventually find, the optimum.

Recombination or sexual reproduction is a key operator for natural evolution. Technically, it takes two genotypes and it produces a new genotype by mixing the gene found in the originals. In biology, the most common form of recombination is crossover: two chromosomes are cut at one point and the halves are spliced to create new chromosomes. The effect of recombination is very important because it allows characteristics from two different parents to be assorted. If the father and the mother possess different good qualities, we would expect that all the good qualities will be passed to the child. Thus the offspring, just by combining all

the good features from its parents, may surpass its ancestors. Many people believe that this mixing of genetic material via sexual reproduction is one of the most powerful features of GAs. As a quick parenthesis about sexual reproduction, GA representation usually does not differentiate male and female individuals (without any perversity). As in many livings species (e.g., snails) any individual can be either a male or a female. In fact, for almost all recombination operators, mother and father are interchangeable.

Mutation is the other way to get new genomes. Mutation consists in changing the value of genes. In natural evolution, mutation mostly engenders non-viable genomes. Actually mutation is not a very frequent operator in natural evolution. Nevertheless, in optimization, a few random changes can be a good way of exploring the search space quickly.

Through those low-level notions of genetic, we have seen how living beings store their characteristic information and how this information can be passed into their offspring. It very basic but it is more than enough to understand the GA theory.

Darwin was totally unaware of the biochemical basics of genetics. Now we know how the genetic inheritable information is coded in DNA, RNA, and proteins and that the coding principles are actually digital, much resembling the information storage in computers. Information processing is in many ways totally different, however. The magnificent phenomenon called the evolution of species can also give some insight into information processing methods and optimization, in particular. According to Darwinism, inherited variation is characterized by the following properties:

1. Variation must be copying because selection does not create directly anything, but presupposes a large population to work on.
2. Variation must be small-scaled in practice. Species do not appear suddenly.
3. Variation is undirected. This is also known as the blind watch maker paradigm.

While the natural sciences approach to evolution has for over a century been to analyze and study different aspects of evolution to find the underlying principles, the engineering sciences are happy to apply evolutionary principles, that have been heavily tested over billions of years, to attack the most complex technical problems, including protein folding.

15.5 Genetic Algorithm vs. Traditional Algorithms

The principle of GAs is simple: imitate genetics and natural selection by a computer program: The parameters of the problem are coded most naturally as a DNA – like linear data structure, a vector or a string. Sometimes, when the problem is naturally two or three dimensional, corresponding array structures are used.

A set, called *population*, of these problem-dependent parameter value vectors is processed by GA. To start, there is usually a totally random population, the values of different parameters generated by a random number generator. Typical population size is from few dozens to thousands. To do optimization we need a cost function or fitness function as it is usually called when GAs are used. By a fitness function we can select the best solution candidates from the population and delete the not so good specimens.

The nice thing when comparing GAs to other optimization methods is that the fitness function can be nearly anything that can be evaluated by a computer or even something that cannot! In the latter case it might be a human judgment that cannot be stated as a crisp program, like in the case of eye witness, where a human being selects from the alternatives generated by GA. So, there are not any definite mathematical restrictions on the properties of the fitness function. It may be discrete, multimodal, etc.

The main criteria used to classify optimization algorithms are as follows: continuous/discrete, constrained/unconstrained and sequential/parallel. There is a clear difference between discrete and continuous problems. Therefore, it is instructive to notice that continuous methods are sometimes used to solve inherently discrete problems and vice versa. Parallel algorithms are usually used to speed up processing. There are, however, some cases in which it is more efficient to run several processors in parallel rather than sequentially. These cases include among others those in which there is high probability of each individual search run to get stuck into a local extreme.

Irrespective of the above classification, optimization methods can be further classified into deterministic and non-deterministic methods. In addition, optimization algorithms can be classified as local or global. In terms of energy and entropy local search corresponds to entropy while global optimization depends essentially on the fitness, i.e., energy landscape.

GA differs from conventional optimization techniques in following ways:

1. GAs operate with coded versions of the problem parameters rather than parameters themselves, i.e., GA works with the coding of solution set and not with the solution itself.
2. Almost all conventional optimization techniques search from a single point, but GAs always operate on a whole population of points (strings), i.e., GA uses population of solutions rather than a single solution for searching. This plays a major role to the robustness of GAs. It improves the chance of reaching the global optimum and also helps in avoiding local stationary point.
3. GA uses fitness function for evaluation rather than derivatives. As a result, they can be applied to any kind of continuous or discrete optimization problem. The key point to be performed here is to identify and specify a meaningful decoding function.
4. GAs use probabilistic transition operates while conventional methods for continuous optimization apply deterministic transition operates, i.e., GAs do not use deterministic rules.

These are the major differences that exist between GA and conventional optimization techniques.

15.6 Basic Terminologies in Genetic Algorithm

The two distinct elements in the GA are individuals and populations. An individual is a single solution while the population is the set of individuals currently involved in the search process.

15.6.1 Individuals

An individual is a single solution. An individual groups together two forms of solutions as given below:

1. The chromosome which is the raw "genetic" information (genotype) that the GA deals.
2. The phenotype which is the expressive of the chromosome in the terms of the model.

A chromosome is subdivided into genes. A gene is the GA's representation of a single factor for a control factor. Each factor in the solution set corresponds to a gene in the chromosome. Figure 15-9 shows the representation of a genotype.

A chromosome should in some way contain information about the solution that it represents. The morphogenesis function associates each genotype with its phenotype. It simply means that each chromosome must define one unique solution, but it does not mean that each solution is encoded by exactly one chromosome. Indeed, the morphogenesis function is not necessarily bijective, and it is even sometimes impossible (especially with binary representation). Nevertheless, the morphogenesis function should at least be subjective. Indeed,

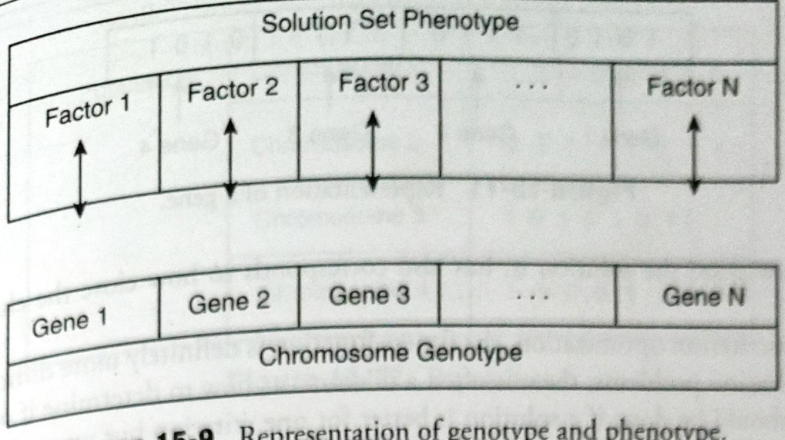


Figure 15-9 Representation of genotype and phenotype.

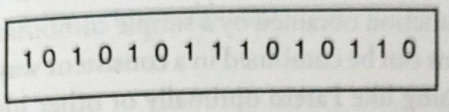


Figure 15-10 Representation of a chromosome.

all the candidate solutions of the problem must correspond to at least one possible chromosome, to be sure that the whole search space can be explored. When the morphogenesis function that associates each chromosome to one solution is not injective, i.e., different chromosomes can encode the same solution, the representation is said to be degenerated. A slight degeneracy is not so worrying, even if the space where the algorithm is looking for the optimal solution is inevitably enlarged. But a too important degeneracy could be a more serious problem. It can badly affect the behavior of the GA, mostly because if several chromosomes can represent the same phenotype, the meaning of each gene will obviously not correspond to a specific characteristic of the solution. It may add some kind of confusion in the search. Chromosomes encoded by bit strings are given in Figure 15-10.

15.6.2 Genes

Genes are the basic "instructions" for building a GA. A chromosome is a sequence of genes. Genes may describe a possible solution to a problem, without actually being the solution. A gene is a bit string of arbitrary lengths. The bit string is a binary representation of number of intervals from a lower bound. A gene is the GA's representation of a single factor value for a control factor, where control factor must have an upper bound and a lower bound. This range can be divided into the number of intervals that can be expressed by the gene's bit string. A bit string of length "n" can represent $(2^n - 1)$ intervals. The size of the interval would be (range)/$(2^n - 1)$.

The structure of each gene is defined in a record of phenotyping parameters. The phenotype parameters are instructions for mapping between genotype and phenotype. It can also be said as encoding a solution set into a chromosome and decoding a chromosome to a solution set. The mapping between genotype and phenotype is necessary to convert solution sets from the model into a form that the GA can work with, and for converting new individuals from the GA into a form that the model can evaluate. In a chromosome, the genes are represented as shown in Figure 15-11.

15.6.3 Fitness

The fitness of an individual in a GA is the value of an objective function for its phenotype. For calculating fitness, the chromosome has to be first decoded and the objective function has to be evaluated. The fitness

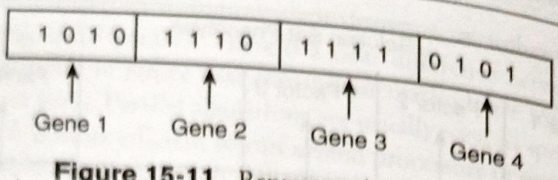


Figure 15-11 Representation of a gene.

not only indicates how good the solution is, but also corresponds to how close the chromosome is to the optimal one.

In the case of multicriterion optimization, the fitness function is definitely more difficult to determine. In multicriterion optimization problems, there is often a dilemma as how to determine if one solution is better than another. What should be done if a solution is better for one criterion but worse for another? But here the trouble comes more from the definition of a "better" solution rather than from how to implement a GA to resolve it. If sometimes a fitness function obtained by a simple combination of the different criteria can give good result, it supposes that criterions can be combined in a consistent way. But, for more advanced problems it may be useful to consider something like Pareto optimally or other ideas from multicriteria optimization theory.

15.6.4 Populations

A population is a collection of individuals. A population consists of a number of individuals being tested, the phenotype parameters defining the individuals and some information about the search space. The two important aspects of population used in GAs are:

1. The initial population generation.
2. The population size.

For each and every problem, the population size will depend on the complexity of the problem. It is often a random initialization of population. In the case of a binary coded chromosome this means that each bit is initialized to a random 0 or 1. However, there may be instances where the initialization of population is carried out with some known good solutions.

Ideally, the first population should have a gene pool as large as possible in order to be able to explore the whole search space. All the different possible alleles of each should be present in the population. To achieve this, the initial population is, in most of the cases, chosen randomly. Nevertheless, sometimes a kind of heuristic can be used to seed the initial population. Thus, the mean fitness of the population is already high and it may help the GA to find good solutions faster. But for doing this one should be sure that the gene pool is still large enough. Otherwise, if the population badly lacks diversity, the algorithm will just explore a small part of the search space and never find global optimal solutions.

The size of the population raises few problems too. The larger the population is, the easier it is to explore the search space. However, it has been established that the time required by a GA to converge is $O(n \log n)$ function evaluations where n is the population size. We say that the population has converged when all the individuals are very much alike and further improvement may only be possible by mutation. Goldberg has also shown that GA efficiency to reach global optimum instead of local ones is largely determined by the size of the population. To sum up, a large population is quite useful. However, it requires much more computational cost memory and time. Practically, a population size of around 100 individuals is quite frequent, but anyway this size can be changed according to the time and the memory disposed on the machine compared to the quality of the result to be reached.

Population	Chromosome 1	1 1 1 0 0 0 1 0
	Chromosome 2	0 1 1 1 1 0 1 1
	Chromosome 3	1 0 1 0 1 0 1 0
	Chromosome 4	1 1 0 0 1 1 0 0

Figure 15-12 Population.

Population being combination of various chromosomes is represented as in Figure 15-12. Thus the population in Figure 15-12 consists of four chromosomes.

15.7 Simple GA

GA handles a population of possible solutions. Each solution is represented through a chromosome, which is just an abstract representation. Coding all the possible solutions into a chromosome is the first part, but certainly not the most straightforward one of a GA. A set of reproduction operators has to be determined, too. Reproduction operators are applied directly on the chromosomes, and are used to perform mutations and recombinations over solutions of the problem. Appropriate representation and reproduction operators are the determining factors, as the behavior of the GA is extremely dependent on it. Frequently, it can be extremely difficult to find a representation that respects the structure of the search space and reproduction operators that are coherent and relevant according to the properties of the problems.

The simple form of GA is given by the following.

1. Start with a randomly generated population.
2. Calculate the fitness of each chromosome in the population.
3. Repeat the following steps until n offsprings have been created:
   - Select a pair of parent chromosomes from the current population.
   - With probability p_c crossover the pair at a randomly chosen point to form two offsprings.
   - Mutate the two offsprings at each locus with probability p_m.
4. Replace the current population with the new population.
5. Go to step 2.

Now we discuss each iteration of this process.

Generation: Selection: is supposed to be able to compare each individual in the population. Selection is done by using a fitness function. Each chromosome has an associated value corresponding to the fitness of the solution it represents. The fitness should correspond to an evaluation of how good the candidate solution is. The optimal solution is the one which maximizes the fitness function. GAs deal with the problems that maximize the fitness function. But, if the problem consists of minimizing a cost function, the adaptation is quite easy. Either the cost function can be transformed into a fitness function, for example by inverting it; or the selection can be adapted in such way that they consider individuals with low evaluation functions as better. Once the reproduction and the fitness function have been properly defined, a GA is evolved according to the same basic structure. It starts by generating an initial population of chromosomes. This first population must

offer a wide diversity of genetic materials. The gene pool should be as large as possible so that any solution of the search space can be engendered. Generally, the initial population is generated randomly. Then, the GA loops over an iteration process to make the population evolve. Each iteration consists of the following steps:

1. *Selection*: The first step consists in selecting individuals for reproduction. This selection is done randomly with a probability depending on the relative fitness of the individuals so that best ones are often chosen for reproduction rather than the poor ones.
2. *Reproduction*: In the second step, offspring are bred by selected individuals. For generating new chromosomes, the algorithm can use both recombination and mutation.
3. *Evaluation*: Then the fitness of the new chromosomes is evaluated.
4. *Replacement*: During the last step, individuals from the old population are killed and replaced by the new ones.

The algorithm is stopped when the population converges toward the optimal solution.

```
BEGIN /* genetic algorithm*/
  Generate initial population;
  Compute fitness of each individual;
  WHILE NOT finished DO LOOP
    BEGIN
      Select individuals from old generations
        For mating;
      Create offspring by applying
        recombination and/or mutation
        to the selected individuals;
      Compute fitness of the new individuals;
      Kill old individuals to make room for
        new chromosomes and insert
        offspring in the new generalization;
      IF Population has converged
        THEN finishes: = TRUE;
    END
END
```

Genetic algorithms are not too hard to program or understand because they are biological based. An example of a flowchart of a GA is shown in Figure 15-13.

15.8 General Genetic Algorithm

The general GA is as follows:

Step 1: *Create a random initial state*: An initial population is created from a random selection of solutions (which are analogous to chromosomes). This is unlike the situation for symbolic AI systems, where the initial state in a problem is already given.

Step 2: *Evaluate fitness*: A value for fitness is assigned to each solution (chromosome) depending on how close it actually is to solving the problem (thus arriving to the answer of the desired problem). (These "solutions" are not to be confused with "answers" to the problem; think of them as possible characteristics that the system would employ in order to reach the answer.)

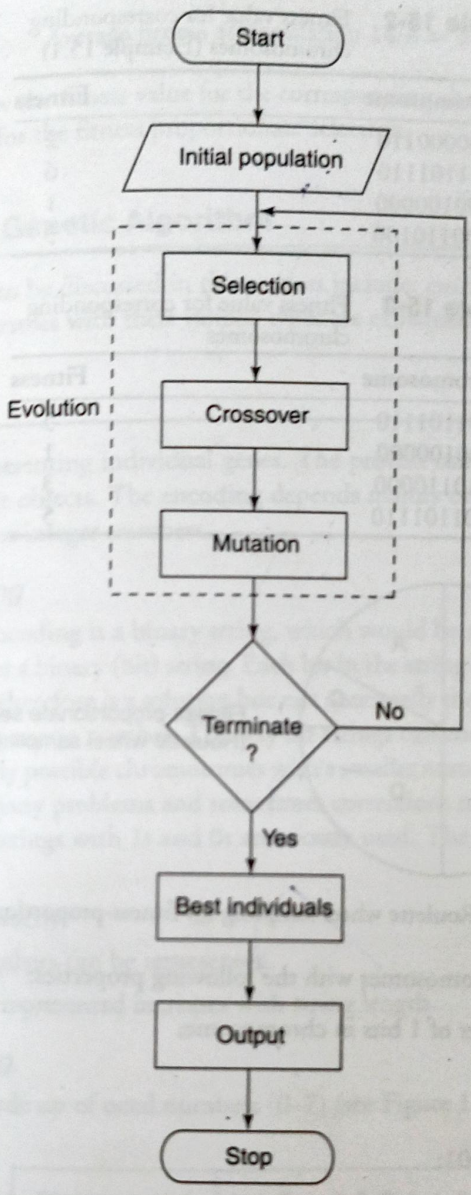


Figure 15-13 Flowchart for genetic algorithm.

Step 3: *Reproduce (and children mutate)*: Those chromosomes with a higher fitness value are more likely to reproduce offspring (which can mutate after reproduction). The offspring is a product of the father and mother, whose composition consists of a combination of genes from the two (this process is known as "crossingover").

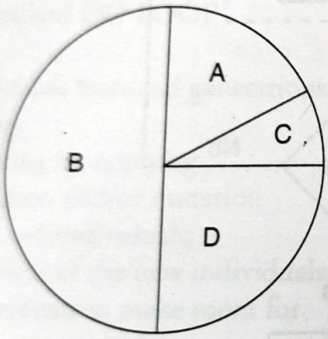
Step 4: *Next generation*: If the new generation contains a solution that produces an output that is close enough or equal to the desired answer then the problem has been solved. If this is not the case, then the new generation will go through the same process as their parents did. This will continue until a solution is reached.

Table 15-2 Fitness value for corresponding chromosomes (Example 15.1)

Chromosome	Fitness
A: 00000110	2
B: 11101110	6
C: 00100000	1
D: 00110100	3

Table 15-3 Fitness value for corresponding chromosomes

Chromosome	Fitness
A: 01101110	5
B: 00100000	1
C: 10110000	3
D: 01101110	5



Fitness-proportionate selection (Roulette wheel sampling)

Figure 15-14 Roulette wheel sampling for fitness-proportionate selection.

Example 15.1: Consider 8-bit chromosomes with the following properties:

1. Fitness function $f(x)$ = number of 1 bits in chromosome;
2. population size $N = 4$;
3. crossover probability $p_c = 0.7$;
4. mutation probability $p_m = 0.001$;

Average fitness of population = 12/4 = 3.0.

1. If B and C are selected, crossover is not performed.
2. If B is mutated, then

$$\text{B: } 11101110 \rightarrow \text{B}'\text{: } 01101110$$

3. If B and D are selected, crossover is performed.

$$\text{B: } 11101110 \text{ E: } 10110100 \rightarrow \text{D: } 00110100 \text{ F: } 01101110$$

4. If E is mutated, then

$$\text{E: } 10110100 \rightarrow \text{E}'\text{: } 10110000$$

Best-fit string from previous population is lost, but the average fitness of population is as given below:

Average fitness of population $14/4 = 3.5$

Tables 15-2 and 15-3 show the fitness value for the corresponding chromosomes and Figure 15-14 shows the Roulette wheel selection for the fitness proportionate selection.

15.9 Operators in Genetic Algorithm

The basic operators that are to be discussed in this section include: encoding, selection, recombination and mutation operators. The operators with their various types are explained with necessary examples.

15.9.1 Encoding

Encoding is a process of representing individual genes. The process can be performed using bits, numbers, trees, arrays, lists or any other objects. The encoding depends mainly on solving the problem. For example, one can encode directly real or integer numbers.

15.9.1.1 Binary Encoding

The most common way of encoding is a binary string, which would be represented as in Figure 15-15.

Each chromosome encodes a binary (bit) string. Each bit in the string can represent some characteristics of the solution. Every bit string therefore is a solution but not necessarily the best solution. Another possibility is that the whole string can represent a number. The way bit strings can code differs from problem to problem.

Binary encoding gives many possible chromosomes with a smaller number of alleles. On the other hand, this encoding is not natural for many problems and sometimes corrections must be made after genetic operation is completed. Binary coded strings with 1s and 0s are mostly used. The length of the string depends on the accuracy. In such coding

1. Integers are represented exactly.
2. Finite number of real numbers can be represented.
3. Number of real numbers represented increases with string length.

15.9.1.2 Octal Encoding

This encoding uses string made up of octal numbers (0–7) (see Figure 15-16).

Chromosome 1	1 1 0 1 0 0 0 1 1 0 1 0
Chromosome 2	0 1 1 1 1 1 1 1 1 1 0 0

Figure 15-15 Binary encoding.

Chromosome 1	03467216
Chromosome 2	15723314

Figure 15-16 Octal encoding.

Chromosome 1	9CE7
Chromosome 2	3DBA

Figure 15-17 Hexadecimal encoding.

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

Figure 15-18 Permutation encoding.

15.9.1.3 Hexadecimal Encoding

This encoding uses string made up of hexadecimal numbers (0–9, A–F) (see Figure 15-17).

15.9.1.4 Permutation Encoding (Real Number Coding)

Every chromosome is a string of numbers, represented in a sequence. Sometimes corrections have to be done after genetic operation is complete. In permutation encoding, every chromosome is a string of integer/real values, which represents number in a sequence.

Permutation encoding (Figure 15-18) is only useful for ordering problems. Even for this problem, some types of crossover and mutation corrections must be made to leave the chromosome consistent (i.e., have real sequence in it).

15.9.1.5 Value Encoding

Every chromosome is a string of values and the values can be anything connected to the problem. This encoding produces best results for some special problems. On the other hand, it is often necessary to develop new genetic operator's specific to the problem. Direct value encoding can be used in problems, where some complicated values, such as real numbers, are used. Use of binary encoding for this type of problems would be very difficult.

In value encoding (Figure 15-19), every chromosome is a string of some values. Values can be anything connected to problem, form numbers, real numbers or characters to some complicated objects. Value encoding is very good for some special problems. On the other hand, for this encoding it is often necessary to develop some new crossover and mutation specific for the problem.

Chromosome A	1.2324 5.3243 0.4556 2.3293 2.4545
Chromosome B	ABDJEIFJDHDIERJFDLDFLFEGT
Chromosome C	(back), (back), (right), (forward), (left)

Figure 15-19 Value encoding.

15.9.1.6 Tree Encoding

This encoding is mainly used for evolving program expressions for genetic programming. Every chromosome is a tree of some objects such as functions and commands of a programming language.

15.9.2 Selection

Selection is the process of choosing two parents from the population for crossing. After deciding on an encoding, the next step is to decide how to perform selection, i.e., how to choose individuals in the population that will create offspring for the next generation and how many offspring each will create. The purpose of selection is to emphasize fitter individuals in the population in hopes that their offspring have higher fitness. Chromosomes are selected from the initial population to be parents for reproduction. The problem is how to select these chromosomes. According to Darwin's theory of evolution the best ones survive to create new offspring. Figure 15-20 shows the basic selection process.

Selection is a method that randomly picks chromosomes out of the population according to their evaluation function. The higher the fitness function, the better chance that an individual will be selected. The selection pressure is defined as the degree to which the better individuals are favored. The higher the selection pressured, the more the better individuals are favored. This selection pressure drives the GA to improve the population fitness over successive generations.

The convergence rate of GA is largely determined by the magnitude of the selection pressure, with higher selection pressures resulting in higher convergence rates. GAs should be able to identify optimal or nearly optimal solutions under a wide range of selection scheme pressure. However, if the selection pressure is too low, the convergence rate will be slow, and the GA will take unnecessarily longer to find the optimal solution. If the selection pressure is too high, there is an increased change of the GA prematurely converging to an incorrect (sub-optimal) solution. In addition to providing selection pressure, selection schemes should also preserve population diversity, as this helps to avoid premature convergence.

Typically we can distinguish two types of selection scheme, proportionate-based selection and ordinal-based selection. Proportionate-based selection picks out individuals based upon their fitness values relative to the fitness of the other individuals in the population. Ordinal-based selection schemes select individuals not upon their raw fitness, but upon their rank within the population. This requires that the selection pressure is independent of the fitness distribution of the population, and is solely based upon the relative ordering (ranking) of the population.

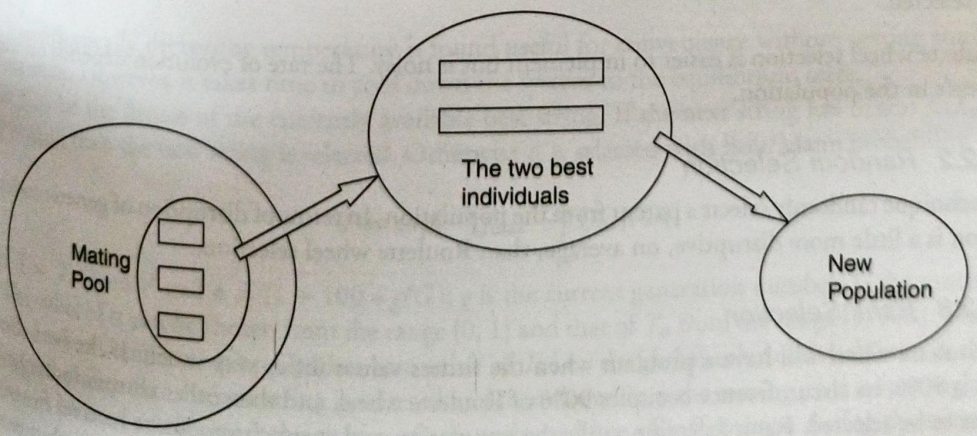


Figure 15-20 Selection.

It is also possible to use a scaling function to redistribute the fitness range of the population in order to adapt the selection pressure. For example, if all the solutions have their fitnesses in the range [999, 1000], the probability of selecting a better individual than any other using a proportionate-based method will not be important. If the fitness every individual is bringing to the range [0, 1] equitable, the probability of selecting good individual instead of bad one will be important.

Selection has to be balanced with variation from crossover and mutation. Too strong selection means sub-optimal highly fit individuals will take over the population, reducing the diversity needed for change and progress; too weak selection will result in too slow evolution. The various selection methods are discussed in the following subsections.

15.9.2.1 Roulette Wheel Selection

Roulette selection is one of the traditional GA selection techniques. The commonly used reproduction operator is the proportionate reproductive operator where a string is selected from the mating pool with a probability proportional to the fitness. The principle of Roulette selection is a linear search through a Roulette wheel with the slots in the wheel weighted in proportion to the individual's fitness values. A target value is set, which is a random proportion of the sum of the fit nesses in the population. The population is stepped through until the target value is reached. This is only a moderately strong selection technique, since fit individuals are not guaranteed to be selected for, but somewhat have a greater chance. A fit individual will contribute more to the target value, but if it does not exceed it, the next chromosome in line has a chance, and it may be weak. It is essential that the population not be sorted by fitness, since this would dramatically bias the selection.

The Roulette process can also be explained as follows: The expected value of an individual is individual's fitness divided by the actual fitness of the population. Each individual is assigned a slice of the Roulette wheel, the size of the slice being proportional to the individual's fitness. The wheel is spun N times, where N is the number of individuals in the population. On each spin, the individual under the wheel's marker is selected to be in the pool of parents for the next generation. This method is implemented as follows:

1. Sum the total expected value of the individuals in the population. Let it be T.
2. Repeat N times:
   i. Choose a random integer "r" between 0 and T.
   ii. Loop through the individuals in the population, summing the expected values, until the sum is greater than or equal to "r." The individual whose expected value puts the sum over this limit is the one selected.

Roulette wheel selection is easier to implement but is noisy. The rate of evolution depends on the variance of fitness's in the population.

15.9.2.2 Random Selection

This technique randomly selects a parent from the population. In terms of disruption of genetic codes, random selection is a little more disruptive, on average, than Roulette wheel selection.

15.9.2.3 Rank Selection

The Roulette wheel will have a problem when the fitness values differ very much. If the best chromosome fitness is 90%, its circumference occupies 90% of Roulette wheel, and then other chromosomes have too few chances to be selected. Rank Selection ranks the population and every chromosome receives fitness from the ranking. The worst has fitness 1 and the best has fitness N. It results in slow convergence but prevents too

quick convergence. It also keeps up selection pressure when the fitness variance is low. It preserves diversity and hence leads to a successful search. In effect, potential parents are selected and a tournament is held to decide which of the individuals will be the parent. There are many ways this can be achieved and two suggestions are:

1. Select a pair of individuals at random. Generate a random number R between 0 and 1. If $R < r$ use the first individual as a parent. If the $R \geq r$ then use the second individual as the parent. This is repeated to select the second parent. The value of r is a parameter to this method.
2. Select two individuals at random. The individual with the highest evaluation becomes the parent. Repeat to find a second parent.

15.9.2.4 Tournament Selection

An ideal selection strategy should be such that it is able to adjust its selective pressure and population diversity so as to fine-tune GA search performance. Unlike, the Roulette wheel selection, the tournament selection strategy provides selective pressure by holding a tournament competition among *Nu* individuals. The best individual from the tournament is the one with the highest fitness, who is the winner of *Nu*. Tournament competitions and the winner are then inserted into the mating pool. The tournament competition is repeated until the mating pool for generating new offspring is filled. The mating pool comprising the tournament winner has higher average population fitness. The fitness difference provides the selection pressure, which drives GA to improve the fitness of the succeeding genes. This method is more efficient and leads to an optimal solution.

15.9.2.5 Boltzmann Selection

SA is a method of function minimization or maximization. This method simulates the process of slow cooling of molten metal to achieve the minimum function value in a minimization problem. Controlling a temperature-like parameter introduced with the concept of Boltzmann probabiiity distribution simulates the cooling phenomenon.

In Boltzmann selection, a continuously varying temperature controls the rate of selection according to a preset schedule. The temperature starts out high, which means that the selection pressure is low. The temperature is gradually lowered, which gradually increases the selection pressure, thereby allowing the GA to narrow in more closely to the best part of the search space while maintaining the appropriate degree of diversity.

A logarithmically decreasing temperature is found useful for convergence without getting stuck to a local minima state. However, it takes time to cool down the system to the equilibrium state.

Let f_{max} be the fitness of the currently available best string. If the next string has fitness $f(X_i)$ such that $f(X_i) > f_{max}$, then the new string is selected. Otherwise it is selected with Bolt/Mann probability

$$P = \exp[-\{f_{max} - f(X_i)\}/T] \tag{15.10}$$

where $T = T_o(1-\alpha)^k$ and $k = (1 + 100 * g/G)$; g is the current generation number; G the maximum value of g. The value of α can be chosen from the range [0, 1] and that of T_o from the range [5, 100]. The final state is reached when computation approaches zero value of T, i.e., the global solution is achieved at this point.

The probability that the best string is selected and introduced into the mating pool is very high. However, Elitism can be used to eliminate the chance of any undesired loss of information during the mutation stage. Moreover, the execution time is less.

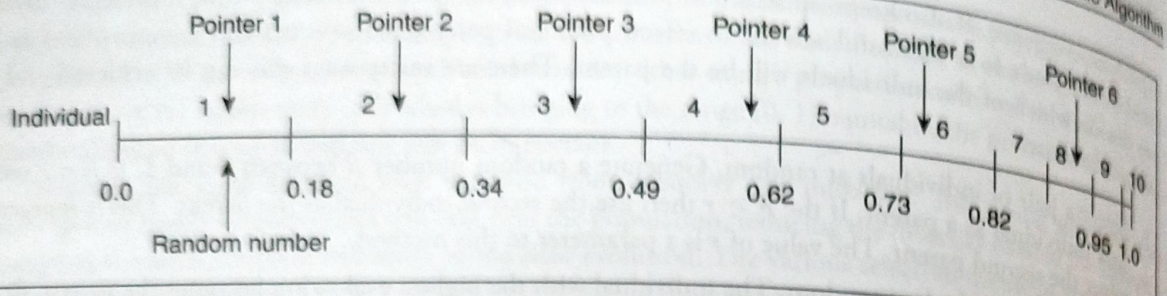


Figure 15-21 Stochastic universal sampling.

litism

he first best chromosome or the few best chromosomes are copied to the new population. The rest is done a classical way. Such individuals can be lost if they are not selected to reproduce or if crossover or mutation estroys them. This significantly improves the GA's performance.

5.9.2.6 *Stochastic Universal Sampling*

tochastic universal sampling provides zero bias and minimum spread. The individuals are mapped to con- guous segments of a line, such that each individual's segment is equal in size to its fitness exactly as in oulette wheel selection. Here equally spaced pointers are placed over the line, as many as there are individ- als to be selected. Consider *NPointer* the number of individuals to be selected, then the distance between he pointers are 1/*NPointer* and the position of the first pointer is given by a randomly generated number in he range [0, 1/*NPointer*]. For 6 individuals to be selected, the distance between the pointers is 1/6 = 0.167. igure 15-21 shows the selection for the above example.

Sample of 1 random number in the range [0, 0.167]: 0.1.
After selection the mating population consists of the individuals,

$$1, 2, 3, 4, 6, 8$$

Stochastic universal sampling ensures selection of offspring that is closer to what is deserved as compared o Roulette wheel selection.

15.9.3 Crossover (Recombination)

Crossover is the process of taking two parent solutions and producing from them a child. After the selection reproduction) process, the population is enriched with better individuals. Reproduction makes clones of ood strings but does not create new ones. Crossover operator is applied to the mating pool with the hope hat it creates a better offspring.

Crossover is a recombination operator that proceeds in three steps:

. The reproduction operator selects at random a pair of two individual strings for the mating.
. A cross site is selected at random along the string length.
. Finally, the position values are swapped between the two strings following the cross site.

That is the simplest way how to do that is to choose randomly some crossover point and copy everything efore this point from the first parent and then copy everything after the crossover point from the other arent. The various crossover techniques are discussed in the following subsections.

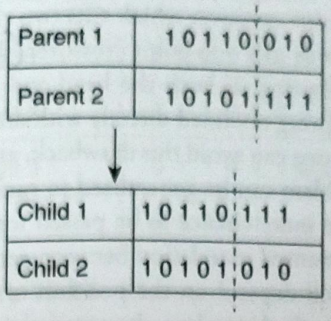


Figure 15-22 Single-point crossover.

15.9.3.1 *Single-Point Crossover*

The traditional genetic algorithm uses single-point crossover, where the two mating chromosomes are cut once at corresponding points and the sections after the cuts exchanged. Here, a cross site or crossover point is selected randomly along the length of the mated strings and bits next to the cross sites are exchanged. If appropriate site is chosen, better children can be obtained by combining good parents, else it severely hampers string quality.

Figure 15-22 illustrates single-point crossover and it can be observed that the bits next to the crossover point are exchanged to produce children. The crossover point can be chosen randomly.

15.9.3.2 *Two-Point Crossover*

Apart from single-point crossover, many different crossover algorithms have been devised, often involving more than one cut point. It should be noted that adding further crossover points reduces the performance of the GA. The problem with adding additional crossover points is that building blocks are more likely to be disrupted. However, an advantage of having more crossover points is that the problem space may be searched more thoroughly.

In two-point crossover, two crossover points are chosen and the contents between these points are exchanged between two mated parents.

In Figure 15-23 the dotted lines indicate the crossover points. Thus the contents between these points are exchanged between the parents to produce new children for mating in the next generation.

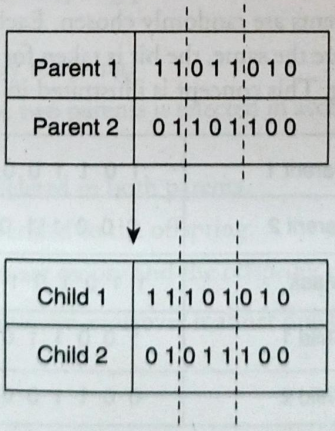


Figure 15-23 Two-point crossover.

Originally, GAs were using one-point crossover which cuts two chromosomes in one point and splices the two halves to create new ones. But with this one-point crossover, the head and the tail of one chromosome cannot be passed together to the offspring. If both the head and the tail of a chromosome contain good genetic information, none of the offspring obtained directly with one-point crossover will share the two good features. Using a two-point crossover one can avoid this drawback, and so it is generally considered better than one-point crossover. In fact, this problem can be generalized to each gene position in a chromosome. Genes that are close on a chromosome have more chance to be passed together to the offspring obtained through *N*-points crossover. It leads to an unwanted correlation between genes next to each other. Consequently, the efficiency of an *N*-point crossover will depend on the position of the genes within the chromosome. In a genetic representation, genes that encode dependent characteristics of the solution should be close together. To avoid all the problem of genes locus, a good thing is to use a uniform crossover as recombination operator.

15.9.3.3 Multipoint Crossover (N-Point Crossover)

There are two ways in this crossover. One is even number of cross sites and the other odd number of cross sites. In the case of even number of cross sites, the cross sites are selected randomly around a circle and information is exchanged. In the case of odd number of cross sites, a different cross point is always assumed at the string beginning.

15.9.3.4 Uniform Crossover

Uniform crossover is quite different from the *N*-point crossover. Each gene in the offspring is created by copying the corresponding gene from one or the other parent chosen according to a random generated binary crossover mask of the same length as the chromosomes. Where there is a 1 in the crossover mask, the gene is copied from the first parent, and where there is a 0 in the mask the gene is copied from the second parent. A new crossover mask is randomly generated for each pair of parents. Offspring, therefore, contain a mixture of genes from each parent. The number of effective crossing point is not fixed, but will average $L/2$ (where L is the chromosome length).

In Figure 15-24, new children are produced using uniform crossover approach. It can be noticed that while producing child 1, when there is a 1 in the mask, the gene is copied from parent 1 else it is copied from parent 2. On producing child 2, when there is a 1 in the mask, the gene is copied from parent 2, and when there is a 0 in the mask, the gene is copied from the parent 1.

15.9.3.5 Three-Parent Crossover

In this crossover technique, three parents are randomly chosen. Each bit of the first parent is compared with the bit of the second parent. If both are the same, the bit is taken for the offspring, otherwise the bit from the third parent is taken for the offspring. This concept is illustrated in Figure 15-25.

Parent 1	1 0 1 1 0 0 1 1
Parent 2	0 0 0 1 1 0 1 0
Mask	1 1 0 1 0 1 1 0
Child 1	1 0 0 1 1 0 1 0
Child 2	0 0 1 1 0 0 1 1

Figure 15-24 Uniform crossover.

Parent 1	1 1 0 1 0 0 0 1
Parent 2	0 1 1 0 1 0 0 1
Parent 3	0 1 1 0 1 1 0 0
Child	0 1 1 0 1 0 0 1

Figure 15-25 Three-parent crossover.

15.9.3.6 Crossover with Reduced Surrogate

The reduced surrogate operator constraints crossover to always produce new individuals wherever possible. This is implemented by restricting the location of crossover points such that crossover points only occur where gene values differ.

15.9.3.7 Shuffle Crossover

Shuffle crossover is related to uniform crossover. A single crossover position (as in single-point crossover) is selected. But before the variables are exchanged, they are randomly shuffled in both parents. After recombination, the variables in the offspring are unshuffled. This removes positional bias as the variables are randomly reassigned each time crossover is performed.

15.9.3.8 Precedence Preservative Crossover

Precedence preservative crossover (PPX) was independently developed for vehicle routing problems by Blanton and Wainwright (1993) and for scheduling problems by Bierwirth et al. (1996). The operator passes on precedence relations of operations given in two parental permutations to one offspring at the same rate, while no new precedence relations are introduced. PPX is illustrated below for a problem consisting of six operations A–F. The operator works as follows:

1. A vector of length Sigma, sub i = 1 to mi, representing the number of operations involved in the problem, is randomly filled with elements of the set {1, 2}.
2. This vector defines the order in which the operations are successively drawn from parent 1 and parent 2.
3. We can also consider the parent and offspring permutations as lists, for which the operations "append" and "delete" are defined.
4. First we start by initializing an empty offspring.
5. The leftmost operation in one of the two parents is selected in accordance with the order of parents given in the vector.
6. After an operation is selected, it is deleted in both parents.
7. Finally the selected operation is appended to the offspring.
8. Step 7 is repeated until both parents are empty and the offspring contains all operations involved.

Note that PPX does not work in a uniform-crossover manner due to the "deletion–append" scheme used. Example is shown in Figure 15-26.

15.9.3.9 Ordered Crossover

Ordered two-point crossover is used when the problem is order based, for example in U-shaped assembly line balancing, etc. Given two parent chromosomes, two random crossover points are selected partitioning

Parent permutation 1	A	B	C	D	E	F
Parent permutation 2	C	A	B	F	D	E
Select parent no. (1/2)	1	2	1	1	2	2
Offspring permutation	A	C	B	D	F	E

Figure 15-26 Precedence preservative crossover (PPX).

Parent 1: 4 2 | 1 3 | 6 5 Child 1: 4 2 | 3 1 | 6 5

Parent 2: 2 3 | 1 4 | 5 6 Child 2: 2 3 | 4 1 | 5 6

Figure 15-27 Ordered crossover.

them into a left, middle and right portions. The ordered two-point crossover behaves in the following way: child 1 inherits its left and right section from parent 1, and its middle section is determined by the genes in the middle section of parent 1 in the order in which the values appear in parent 2. A similar process is applied to determine child 2. This is shown in Figure 15-27.

15.9.3.10 Partially Matched Crossover

Partially matched crossover (PMX) can be applied usefully in the TSP. Indeed, TSP chromosomes are simply sequences of integers, where each integer represents a different city and the order represents the time at which a city is visited. Under this representation, known as permutation encoding, we are only interested in labels and not alleles. It may be viewed as a crossover of permutations that guarantees that all positions are found exactly once in each offspring, i.e., both offspring receive a full complement of genes, followed by the corresponding filling in of alleles from their parents. PMX proceeds as follows:

1. The two chromosomes are aligned.
2. Two crossing sites are selected uniformly at random along the strings, defining a matching section.
3. The matching section is used to effect a cross through position-by-position exchange operation.
4. Alleles are moved to their new positions in the offspring.

The following illustrates how PMX works.

Name 9 8 4 . 5 6 7 . 1 3 2 10 Allele 1 0 1 . 0 0 1 . 1 1 0 0

Name 8 7 1 . 2 3 10 . 9 5 4 6 Allele 1 1 1 . 0 1 1 . 1 1 0 1

Figure 15-28 Given strings.

Consider the two strings shown in Figure 15-28, where the dots mark the selected cross points. The matching section defines the position-wise exchanges that must take place in both parents to produce the offspring. The exchanges are read from the matching section of one chromosome to that of the other. In the example illustrate in Figure 15-28, the numbers that exchange places are 5 and 2, 6 and 3, and 7 and 10. The resulting offspring are as shown in Figure 15-29. PMX is dealt in detail in the next chapter.

Name 984.2310.1657 Allele 101.010.1001

Name 8101.567.9243 Allele 111.111.1001

Figure 15-29 Partially matched crossover.

15.9.3.11 Crossover Probability

The basic parameter in crossover technique is the crossover probability (P_c). Crossover probability is a parameter to describe how often crossover will be performed. If there is no crossover, offspring are exact copies of parents. If there is crossover, offspring are made from parts of both parents' chromosome. If crossover probability is 100%, then all offspring are made by crossover. If it is 0%, whole new generation is made from exact copies of chromosomes from old population (but this does not mean that the new generation is the same!). Crossover is made in hope that new chromosomes will contain good parts of old chromosomes and therefore the new chromosomes will be better. However, it is good to leave some part of old population survive to next generation.

15.9.4 Mutation

After crossover, the strings are subjected to mutation. Mutation prevents the algorithm to be trapped in a local minimum. Mutation plays the role of recovering the lost genetic materials as well as for randomly distributing genetic information. It is an insurance policy against the irreversible loss of genetic material. Mutation has been traditionally considered as a simple search operator. If crossover is supposed to exploit the current solution to find better ones, mutation is supposed to help for the exploration of the whole search space. Mutation is viewed as a background operator to maintain genetic diversity in the population. It introduces new genetic structures in the population by randomly modifying some of its building blocks. Mutation helps escape from local minima's trap and maintains diversity in the population. It also keeps the gene pool well stocked, thus ensuring ergodicity. A search space is said to be ergodic if there is a non-zero probability of generating any solution from any population state.

There are many different forms of mutation for the different kinds of representation. For binary representation, a simple mutation can consist in inverting the value of each gene with a small probability. The probability is usually taken about $1/L$, where L is the length of the chromosome. It is also possible to implement kind of hill climbing mutation operators that do mutation only if it improves the quality of the solution. Such an operator can accelerate the search; however, care should be taken, because it might also reduce the diversity in the population and make the algorithm converge toward some local optima. Mutation of a bit involves flipping a bit, changing 0 to 1 and vice-versa.

15.9.4.1 Flipping

Flipping of a bit involves changing 0 to 1 and 1 to 0 based on a mutation chromosome generated. Figure 15-30 explains mutation-flipping concept. A parent is considered and a mutation chromosome is randomly generated. For a 1 in mutation chromosome, the corresponding bit in parent chromosome is flipped (0 to 1 and 1 to 0) and child chromosome is produced. In the case illustrated in Figure 15-30, 1 occurs at 3 places of mutation chromosome, the corresponding bits in parent chromosome are flipped and the child is generated.

15.9.4.2 Interchanging

Two random positions of the string are chosen and the bits corresponding to those positions are interchanged (Figure 15.31).

Parent	1 0 1 1	0 1 0	1
Mutation chromosome	1 0 0 0	1 0 0	1
Child	0 0 1 1	1 1 0	0

Figure 15-30 Mutation flipping.

Parent	1 0 1 1 0 1 0 1
Child	1 1 1 1 0 0 0 1

Figure 15-31 Interchanging.

Parent	1 0 1 1 0	1 0 1
Child	1 0 1 1 0	1 1 0

Figure 15-32 Reversing.

15.9.4.3 *Reversing*

A random position is chosen and the bits next to that position are reversed and child chromosome is produced (Figure 15-32).

15.9.4.4 *Mutation Probability*

An important parameter in the mutation technique is the mutation probability (P_m). It decides how often parts of chromosome will be mutated. If there is no mutation, offspring are generated immediately after crossover (or directly copied) without any change. If mutation is performed, one or more parts of a chromosome are changed. If mutation probability is 100%, whole chromosome is changed; if it is 0%, nothing is changed. Mutation generally prevents the GA from falling into local extremes. Mutation should not occur very often, because then GA will in fact change to random search.

15.10 Stopping Condition for Genetic Algorithm Flow

In short, the various stopping condition are listed as follows:

1. *Maximum generations*: The GA stops when the specified number of generations has evolved.
2. *Elapsed time*: The genetic process will end when a specified time has elapsed.
   Note: If the maximum number of generation has been reached before the specified time has elapsed, the process will end.
3. *No change in fitness*: The genetic process will end if there is no change to the population's best fitness for a specified number of generations.
   Note: If the maximum number of generation has been reached before the specified number of generation with no changes has been reached, the process will end.

4. *Stall generations*: The algorithm stops if there is no improvement in the objective function for a sequence of consecutive generations of length "Stall generations."
5. *Stall time limit*: The algorithm stops if there is no improvement in the objective function during an interval of time in seconds equal to "Stall time limit."

The termination or convergence criterion finally brings the search to a halt. The following are the few methods of termination techniques.

15.10.1 Best Individual

A best individual convergence criterion stops the search once the minimum fitness in the population drops below the convergence value. This brings the search to a faster conclusion, guaranteeing at least one good solution.

15.10.2 Worst Individual

Worst individual terminates the search when the least fit individuals in the population have fitness less than the convergence criteria. This guarantees the entire population to be of minimum standard, although the best individual may not be significantly better than the worst. In this case, a stringent convergence value may never be met, in which case the search will terminate after the maximum has been exceeded.

15.10.3 Sum of Fitness

In this termination scheme, the search is considered to have satisfaction converged when the sum of the fitness in the entire population is less than or equal to the convergence value in the population record. This guarantees that virtually all individuals in the population will be within a particular fitness range, although it is better to pair this convergence criteria with weakest gene replacement, otherwise a few unfit individuals in the population will blow out the fitness sum. The population size has to be considered while setting the convergence value.

15.10.4 Median Fitness

Here at least half of the individuals will be better than or equal to the convergence value, which should give a good range of solutions to choose from.

15.11 Constraints in Genetic Algorithm

If the GA considered consists of only objective function and no information about the specifications of variable, then it is called *unconstrained optimization problem*. Consider, an unconstrained optimization problem of the form

$$\text{Minimize } f(x) = x^2 \tag{15.11}$$

and there is no information about "x" range. GA minimizes this function using its operators in random specifications.

In the case of constrained optimization problems, the information is provided for the variables under consideration. Constraints are classified as:

1. Equality relations.
2. Inequality relations.

A GA generates a sequence of parameters to be tested using the system under consideration, objective function (to be maximized or minimized) and the constraints. On running the system, the objective function is evaluated and constraints are checked to see if there are any violations. If there are no violations, the parameter set is assigned the fitness value corresponding to the objective function evaluation. When the constraints are violated, the solution is infeasible and thus has no fitness. Many practical problems are constrained and it is very difficult to find a feasible point that is best. As a result, one should get some information out of infeasible solutions, irrespective of their fitness ranking in relation to the degree of constraint violation. This is performed in penalty method.

Penalty method is one where a constrained optimization problem is transformed to an unconstrained optimization problem by associating a penalty or cost with all constraint violations. This penalty is included in the objective function evaluation.

Consider the original constrained problem in maximization form:

$$\text{Maximize } f(x)$$

$$\text{Subject to } g_i(x) \geq 0, \quad i = 1, 2, 3, \ldots, n \tag{15.12}$$

where x is a k-vector. Transforming this to unconstrained form:

$$\text{Maximize } f(x) + P \sum_{i=1}^{n} \Phi [g_i(x)] \tag{15.13}$$

where Φ is the penalty function and P is the penalty coefficient. There exist several alternatives for this penalty function. The penalty function can be squared for all violated constraints. In certain situations, the unconstrained solution converges to the constrained solution as the penalty coefficient p tends to infinity.

15.12 Problem Solving Using Genetic Algorithm

15.12.1 Maximizing a Function

Consider the problem of maximizing the function,

$$f(x) = x^2 \tag{15.14}$$

where x is permitted to vary between 0 and 31. The steps involved in solving this problem are as follows:

Step 1: For using GA approach, one must first code the decision variable "x" into a finite length string. Using a five bit (binary integer) unsigned integer, numbers between 0(00000) and 31(11111) can be obtained.

The objective function here is $f(x) = x^2$ which is to be maximized. A single generation of a GA is performed here with encoding, selection, crossover and mutation. To start with, select initial population at random. Here initial population of size 4 is chosen, but any number of populations can be selected based on the requirement and application. Table 15-4 shows an initial population randomly selected.

Table 15-4 Selection

String no.	Initial population (randomly selected)	x value	Fitness $f(x)=x^2$	Prob$_i$	Percentage probability (%)	Expected count	Actual count
1	01100	12	144	0.1247	12.47	0.4987	1
2	11001	25	625	0.5411	54.11	2.1645	2
3	00101	5	25	0.0216	2.16	0.0866	0
4	10011	19	361	0.3126	31.26	1.2502	1
Sum			1155	1.0000	100	4.0000	4
Average			288.75	0.2500	25	1.0000	1
Maximum			625	0.5411	54.11	2.1645	2

Step 2: Obtain the decoded x values for the initial population generated. Consider string 1,

$$\begin{aligned} 01100 &= 0*2^4 + 1*2^3 + 1*2^2 + 0*2^1 + 0*2^0 \\ &= 0 + 8 + 4 + 0 + 0 \\ &= 12 \end{aligned}$$

Thus for all the four strings the decoded values are obtained.

Step 3: Calculate the fitness or objective function. This is obtained by simply squaring the "x" value, since the given function is $f(x) = x^2$. When $x = 12$, the fitness value is

$$f(x) = x^2 = (12)^2 = 144$$

$$\text{For } x = 25, \quad f(x) = x^2 = (25)^2 = 625$$

and so on, until the entire population is computed.

Step 4: Compute the probability of selection,

$$\text{Prob}_i = \frac{f(x)_i}{\sum_{i=1}^{n} f(x)_i} \tag{15.15}$$

where n is the number of populations; $f(x)$ is the fitness value corresponding to a particular individual in the population;

$\Sigma f(x)$ is the summation of all the fitness value of the entire population.

Considering string 1,

$$\text{Fitness } f(x) = 144$$

$$\Sigma f(x) = 1155$$

The probability that string 1 occurs is given by

$$P_1 = 144/1155 = 0.1247$$

The percentage probability is obtained as

$$0.1247 * 100 = 12.47\%$$

The same operation is done for all the strings. It should be noted that summation of probability select is 1.

Step 5: The next step is to calculate the expected count, which is calculated as

$$\text{Expected count} = \frac{f(x)_i}{[\text{Avg}\, f(x)]_i} \tag{15.16}$$

where

$$(\text{Avg}\, f(x))_i = \left[\frac{\sum_{i=1}^{n} f(x)_i}{n}\right]$$

For string 1,

$$\text{Expected count} = \text{Fitness/Average} = 144/288.75 = 0.4987$$

We then compute the expected count for the entire population. The expected count gives an idea of which population can be selected for further processing in the mating pool.

Step 6: Now the actual count is to be obtained to select the individuals who would participate in the crossover cycle using Roulette wheel selection. The Roulette wheel is formed as shown Figure 15-33.

The entire Roulette wheel covers 100% and the probabilities of selection as calculated in step 4 for the entire populations are used as indicators to fit into the Roulette wheel. Now the wheel may be spun and the number of occurrences of population is noted to get actual count.

1. String 1 occupies 12.47%, so there is a chance for it to occur at least once. Hence its actual count may be 1.
2. With string 2 occupying 54.11% of the Roulette wheel, it has a fair chance of being selected twice. Thus its actual count can be considered as 2.
3. On the other hand, string 3 has the least probability percentage of 2.16%, so their occurrence for next cycle is very poor. As a result, it actual count is 0.

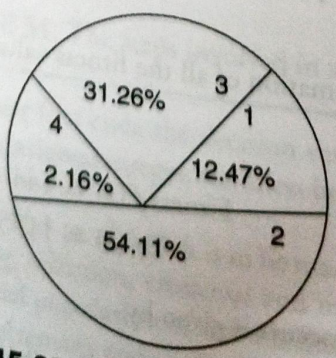


Figure 15-33 Selection using Roulette wheel.

Table 15-5 Crossover

String no.	Mating Pool	Crossover point	Offspring after crossover	x value	Fitness value $f(x) = x^2$
1	0 1 1 | 0 0	4	0 1 1 0 1	13	169
2	1 1 0 | 0 1	4	1 1 0 0 0	24	576
3	1 | 0 0 1	2	1 1 0 1 1	27	729
4	1 | 0 1 1	2	1 0 0 0 1	17	289
Sum					1763
Average					440.75
Maximum					729

4. String 4 with 31.26% has at least one chance for occurring while Roulette wheel is spun, thus its actual count is 1.

The above values of actual count are tabulated as shown is Table 15-5.

Step 7: Now, write the mating pool based upon the actual count as shown in Table 15-5.
The actual count of string no. 1 is 1, hence it occurs once in the mating pool. The actual count of string no. 2 is 2, hence it occurs twice in the mating pool. Since the actual count of string no. 3 is 0, it does not occur in the mating pool. Similarly, the actual count of string no. 4 being 1, it occurs once in the mating pool. Based on this, the mating pool is formed.

Step 8: Crossover operation is performed to produce new offspring (children). The crossover point is specified and based on the crossover point, single-point crossover is performed and new offspring is produced. The parents are

Parent 1	0 1 1 0 0
Parent 2	1 1 0 0 1

The offspring is produced as

Offspring 1	0 1 1 0 1
Offspring 2	1 1 0 0 0

In a similar manner, crossover is performed for the next strings.

Step 9: After crossover operations, new offspring are produced and "x" values are decoded and fitness is calculated.

Step 10: In this step, mutation operation is performed to produce new offspring after crossover operation. As discussed in Section 15.9.4.1 mutation-flipping operation is performed and new offspring are produced. Table 15-6 shows the new offspring after mutation. Once the offspring are obtained after mutation, they are decoded to x value and the fitness values are computed.

This completes one generation. The mutation is performed on a bit-bit by basis. The crossover probability and mutation probability were assumed to be 1.0 and 0.001, respectively. Once selection, crossover and

Table 15-6 Mutation

String no.	Offspring after crossover	Mutation chromosomes for flipping	Offspring after mutation	x value	Fitness $f(x)=x^2$
1	0 1 1 0 1	1 0 0 0 0	1 1 1 0 1	29	841
2	1 1 0 0 0	0 0 0 0 0	1 1 0 0 0	24	576
3	1 1 0 1 1	0 0 0 0 0	1 1 0 1 1	27	729
4	1 0 0 0 1	0 0 1 0 0	1 0 1 0 0	20	400
Sum					2546
Average					636.5
Maximum					841

mutation are performed, the new population is now ready to be tested. This is performed by decoding the new strings created by the simple GA after mutation and calculates the fitness function values from the x values thus decoded. The results for successive cycles of simulation are shown in Tables 15-4 and 15-6.

From the tables, it can be observed how GAs combine high-performance notions to achieve better performance. In the tables, it can be noted how maximal and average performances have improved in the new population. The population average fitness has improved from 288.75 to 636.5 in one generation. The maximum fitness has increased from 625 to 841 during the same period. Although random processes make this best solution, its improvement can also be seen successively. The best string of the initial population (1 1 0 0 1) receives two chances for its existence because of its high, above-average performance. When this combines at random with the next highest string (1 0 0 1 1) and is crossed at crossover point 2 (as shown in Table 15-5), one of the resulting strings (1 1 0 1 1) proves to be a very best solution indeed. Thus after mutation at random, a new offspring (1 1 1 0 1) is produced which is an excellent choice.

This example has shown one generation of a simple GA.

15.13 The Schema Theorem

In this section, we will formulate and prove the fundamental result on the behavior of GAs – the so-called Schema Theorem. Although being completely incomparable with convergence results for conventional optimization methods, it still provides valuable insight into the intrinsic principles of GAs. Assume a GA with proportional selection and an arbitrary but fixed fitness function f. Let us make the following notations:

1. The number of individuals which fulfill H at time step t are denoted as

$$r_{H,t} = |B_t \cap H|$$

2. The expression $\bar{f}(t)$ refers to the observed average fitness at time t:

$$\bar{f}(t) = \frac{1}{m}\sum_{i=1}^{m} f(b_{i,t})$$

3. The term $\bar{f}(H,t)$ stands for the observed average fitness of schema H in time step t:

$$\bar{f}(H,t) = \frac{1}{r_{H,t}} \sum_{i \in \{j | b_{j,t} \in H\}} f(b_{i,t})$$

Theorem (Schema Theorem – Holland 1975). Assuming we consider a simple GA, the following inequality holds for every schema H:

$$E[r_{H,t+1}] \geq r_{H,t} \frac{\bar{f}(H,t)}{\bar{f}(t)} \left(1 - p_c \frac{\delta(H)}{n-1}\right) (1-p_M)^{O(H)}$$

Proof. The probability that we select an individual fulfilling H is

$$\frac{\sum_{i \in \{j | b_{j,t} \in H\}} f(b_{i,t})}{\sum_{i=1}^{m} f(b_{i,t})}$$

This probability does not change throughout the execution of the selection loop. Moreover, each of the m individuals is selected independent of the others. Hence, the number of selected individuals, which fulfill H, is binomially distributed with sample amount m and the probability. We obtain, therefore, that the expected number of selected individuals fulfilling H is

$$m \frac{\sum_{i \in \{j | b_{j,t} \in H\}} f(b_{i,t})}{\sum_{i=1}^{m} f(b_{i,t})} = m \frac{r_{H,t}}{r_{H,t}} \frac{\sum_{i \in \{j | b_{j,t} \in H\}} f(b_{i,t})}{\sum_{i=1}^{m} f(b_{i,t})} = r_{H,t} \frac{\sum_{i \in \{j | b_{j,t} \in H\}} f(b_{i,t})/r_{H,t}}{\sum_{i=1}^{m} f(b_{i,t})/m} \leq r_{H,t} \frac{\bar{f}(H,t)}{\bar{f}(t)}$$

If two individuals are crossed, which both fulfill H, the two offsprings again fulfill H. The number of strings fulfilling H can only decrease if one string, which fulfills H, is crossed with a string which does not fulfill H, but, obviously, only if the cross site is chosen somewhere in between the specifications of H. The probability that the cross site is chosen within the defining length of H is

$$\frac{\delta(H)}{n-1}$$

Hence the survival probability p_S of H, i.e., the probability that a string fulfilling H produces an offspring also fulfilling H, can be estimated as follows (crossover is only done with probability p_C):

$$p_S \geq 1 - p_C \frac{\delta(H)}{n-1}$$

Selection and crossover are carried out independently, so we may compute the expected number of strings fulfilling H after crossover simply as

$$\frac{\bar{f}(H,t)}{\bar{f}(t)} r_{H,t}\, p_S \geq \frac{\bar{f}(H,t)}{\bar{f}(t)} r_{H,t} \left(1 - p_C \frac{\delta(H)}{n-1}\right)$$

After crossover, the number of strings fulfilling H can only decrease if a string fulfilling H is altered by mutation at a specification of H. The probability that all specifications of H remain untouched by mutation is obviously

$$(1-p_M)^{O(H)}$$

The arguments in the proof of the Schema Theorem can be applied analogously to many other crossover and mutation operations.