IPMP Journal of Management & Science (IPMP-JMS), ISSN: 2590-9681, Scientific Journal Impact Factor (SJIF): 4.309
Vol 2, Issue 06, June, 2018, Pages 93 - 130, Available @ www.ipmp-jms.org

Working Principles of Genetic Algorithm

Andrew Owusu-Hemeng[1], Peter Kwasi Sarpong[2], Joseph Ackora-Prah[3]
[1,2,3]Department of Mathematics, Kwame Nkrumah University of Science and Technology,Kumasi,Ghana
Email: owusuhemengandrew@gmail.com, kp.sarp@yahoo.co.uk, ackph@yahoo.co.uk

Abstract
Genetic Algorithms are a family of computational models inspired by evolution or Genetic Algorithms are search algorithms that are based on concepts of natural selection and natural genetics. Thus GA is a stochastic global search method that mimics the metaphor of natural biological evolution. These algorithms encode a potential solution to a specific problem on a simple chromosome-like data structure and apply recombination operators to these structures so as to preserve critical information. Genetic algorithm was developed to simulate some of the processes observed in natural evolution, a process that operates on chromosomes (organic devices for encoding the structure of living being). GAs operate on a number of potential solutions, called a population, consisting of some encoding of the parameter set simultaneously and applying the principle of survival of the fittest to produce (hopefully) better and better approximations to a solution. Genetic algorithms are often viewed as function optimizer, although the range of problems to which genetic algorithms have been applied are quite broad. An implementation of genetic algorithm begins with a population of (typically random) chromosomes. A new population is created by allowing parent solutions in one generation to produce offspring, which are included in the next generation. A 'survival of the fittest' principle is applied to ensure that the overall quality of solutions increases as the algorithm progresses from one generation to the next. *The overall structure of genetic algorithm is as follows Gen and Cheng (200), Goldberg (1989)): Selection & Crossover*

I. MUTATION

In GA, each individual represents a potential solution to the problem at hand. Each individual is evaluated to give some measure of its fitness. Some individuals undergo stochastic transformations by means of genetic operations to form new individuals. There are two type of transformation:-

- Crossover, which creates new individuals by combining parts from two individuals.
- Mutation, which creates new individuals by making changes in a single individual.

A general framework and a possible implementation of a genetic algorithm for a permutation scheduling problem is given below.

- ***Initialization***- Choose initial population P containing q solutions to be the current population (randomly generate q permutations also called *strings*).
- ***Evaluation***- Compute a fitness value for each solution of P (compute the value $F(S)$ for each solution S).
- ***Reproduction***- Use fitness values to select solutions from P to form a mating pool (select $q/2$ best permutations).
- ***Regeneration***- Apply *crossover*, *mutation* and any other selected operations to solutions of the mating pool to form a new population ($q/2$ new permutations are obtained and replace $q/2$ worst permutations in the population).
- ***Termination test***- Test whether the algorithm should terminate. If it terminates, output the best solution generated; otherwise, return to the evaluation step.

In the initialization step, we create a population by generating q random permutations. A non-negative fitness function $F(S)$ is used in the evaluation step. In the reproduction step, a mating pool of size $q/2$ is created. To apply the crossover operation in the regeneration step, solutions in the mating pool are randomly partitioned into pairs. With probability $p$cross, each pair undergoes a crossover; otherwise, the pair is unchanged. Under a crossover operation, the two solutions, which we refer to as *parents*, combine to produce two *offspring*, each containing some characteristics of each parent. The hope is that one of the offspring will inherit the desirable features of each parent to produce a good quality solution. A mutation operation is applied to solutions before placing them into the new population, each element of each string (each jobs in the permutation) is selected with probability $p$mut to be perturbed. E.g., if a job of a string is selected for mutation, then it is swapped with another randomly selected job in the same string (which yields a neighbour in the swap neighbourhood). As a termination test, a time limit is set and the algorithm terminates when this limit is exceeded. Thus the major steps involved are the generation of a population of solutions, finding the objective function and fitness function and the application of genetic operators. These aspects are described briefly below.

```
Begin GA
g = 0 {generation counter}
Initiation population $P(g)$
Evaluation  population $P(g)$ {i.e., compute fitness value}
While not done do
$g = g + 1$
Select $P(g)$ from $P(g-1)$
Crossover $P(g)$
Mutate $P(g)$
Evaluate  $P(g)$
end
end GA
```

For the basic GA operations: One generation is broken down into a selection phase and recombination phase. Strings are assigned into adjacent slots during selection. An important characteristic of genetic algorithm is the coding of variables that describes the problem. The most common coding method is to transform the variables to a binary string or vector; GAs perform best when solution vectors are binary. If the problem has more than one variable, a multi-variable coding is constructed by concatenating as many single variables coding as the number of variables in the problem. Genetic Algorithm processes a number of solutions simultaneously. At each generation, a new set of approximations is created by the process of selecting individuals according to their level of fitness in the problem domain and breeding them together using operators borrowed from natural genetics. This process leads to the evolution of populations of individuals that are better suited to their environment than the individuals that they were created from, just as in natural adaptation. The genetic algorithm deifiers from other search methods in that it searches among a population of points, and works with a coding of parameter set, rather than the parameter values themselves. The transition scheme of the genetic algorithm is probabilistic, whereas traditional methods use gradient information. Because of these features of genetic algorithm, they are used as general purpose optimization algorithm. They also provide means to search irregular space and hence are applied to a variety of function optimization, parameter estimation and machine learning applications. On the assumption that the initial population is generated by a random sample with replacement (which is a conservative assumption in this context), the probability that at least one allele is present at each locus can be found. For binary strings this is easily seen to be from which we can calculate that, for example, a population of size 17 is enough to ensure that the required probability exceeds 99.9% for strings of length 50.

Genetic algorithms work on two types of spaces alternatively: Coding space and solution space, or in other words, genotype space and phenotype space. Genetic operators (crossover and mutation) work on genotype space, while evolution and selection work on phenotype space. The selection is the link between chromosomes and the performance of decoded solutions. The mapping from genotype space to phenotype space has a considerable influence on the performance of genetic algorithms. The genetic algorithms provide a directed random search in complex landscapes. There are two important issues with respect to search strategies: exploration (investigate new and unknown areas in search space) and exploitation (make use of knowledge of solutions previously found in search space to help in find better solutions). This can be done by making genetic operators perform essentially a blind search; with a hope that selection operators direct the genetic search toward the desirable area of solution space. One general principle for developing an implementation of genetic algorithms for a particular real word problem is to make a good balance between exploration and exploitation of the search space. To achieve this, all the operators and parameters of the genetic algorithms must be examined carefully. Addition heuristics should be incorporated in the algorithm to enhance the performance. Unlike simple neighborhood search methods that terminate when a local optimum is reached, GAs are stochastic search methods that could in principle run for ever. In practice, a termination criterion is needed; common approaches are to set a limit on the number of fitness evaluations or the computer clock time, or to track the population's diversity and stop when this falls below a preset threshold. The meaning of diversity in the latter case is not always obvious, and it could relate either to the genotype or the phenotype, or even, conceivably, to the fitnesses, but the most common way to measure it is by genotype statistics. For example, we could decide to terminate a run if at every locus the proportion of one particular allele rose above 90%. After several generations, genetic algorithm converges to the best individual, which hopefully represents an optimal or suboptimal solution to the problem.

II. ENCODING

As for any search and learning method, the way in which candidate solutions are encoded is a central, if not *the* central, factor in the success of a genetic algorithm. Encoding is a process performed using bits, arrays, trees, numbers or list to represent individual genes. Most GA applications use fixed−length, fixed−order bit strings to encode candidate solutions. How to encode the solutions of the problem into chromosomes is a key issue when using genetic algorithms. One outstanding problem associated with encoding is that some individuals correspond to infeasible or illegal solutions to a given problem. This may become very severe for constrained optimization problems and combinatorial optimization problems. It must be distinguished between two concepts: Infeasibility and Illegality. Infeasibility refers to the phenomenon that a solution decoded from chromosome lies outside the feasible region of given problem. Penalty methods can be used to handle infeasible chromosomes. One of these methods is by force genetic algorithms to approach optimal form both sides of feasible and infeasible regions. Illegality refers to the phenomenon that a chromosome does not represent a solution to a given problem. Repair techniques are usually adopted to convert an illegal chromosome to legal one.

Encoding can be adapted and one reason for adapting the encoding is that a fixed−length representation limits the complexity of the candidate solutions. For example, in the Prisoner's Dilemma example, Axelrod fixed the memory of the evolving strategies to three games, requiring a chromosome of length 64 plus a few extra bits to encode initial conditions. But it would be interesting to know what types of strategies could evolve if the memory size were allowed to increase or decrease (requiring variable−length chromosomes). Various encoding methods have been created for particular problems to provide effective implementation of genetic algorithms. According to what kind of symbol is used as the alleles of a gene, the encoding methods can be classified as follows:

- Binary encoding
- Valued encoding
- Permutation encoding
- Tree encoding
- Octal Encoding

A. Binary Encodings

Binary encodings (i.e., bit strings) are the most common encodings for a number of reasons. One is historical: in their earlier work, Holland and his students concentrated on such encodings and GA practice has tended to follow this lead. In Binary-coded strings having 1's and 0's are mostly used. Thus the data value is converted into binary strings. It gives many chromosomes with small number of alleles. The length of the string is usually determined according to the desired solution accuracy. Much of the existing GA theory is based on the assumption of fixed−length, fixed−order binary encodings. Much of that theory can be extended to apply to non binary encodings, but such extensions are not as well developed as the original theory. A chromosome represented in a binary encoding is shown in Fig. 3.1

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

Fig 3.1 Binary Encoding

B. Valued Encodings

For many applications, it is most natural to use an alphabet of many characters or real numbers to form chromosomes. Valued encoding is best used for function optimization problems .It is often required to develop new GA operators

specific for the problem in valued encoding. It has been widely confirmed that valued encoding perform better than binary encoding for function optimization and constrained optimizations problems. Examples include Kitano's many−character representation for graph−generation grammars, Meyer and Packard's real−valued representation for condition sets, Montana and Davis's real−valued representation for neural−network weights, and Schultz−Kremer's real−valued representation for torsion angles in proteins.
Holland's schema−counting argument seems to imply that GAs should exhibit worse performance on valued encoding than on binary encodings. Several empirical comparisons between binary encodings and valued encodings have shown better performance for the latter. Valued encoding can be seen in Fig. 3.02 below

Chromosome 1	6.5434 1.7543 0.0012 2.9112 5.0654
Chromosome 2	Left right back forward center

Fig. 3.2 Value Encoding

C. Permutation Encoding
Permutation encoding is best used for combinational optimization problems because the essence of this kind of problems is to search for the best permutation or combination of items subject to constrains. In permutation encoding, every chromosome is a string of numbers which represents number in a sequence as shown below in Fig. 3.3

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

Fig. 3.3 Permutation Encoding

D. Tree Encoding
In tree encoding, every chromosome is a tree of some objects, functions or commands in programming languages. Tree encoding schemes, such as John Koza's scheme for representing computer programs, have several advantages, including the fact that they allow the search space to be open−ended (in principle, any size tree could be formed via crossover and mutation). This open−endedness also leads to some potential pitfalls. The trees can grow large in uncontrolled ways, preventing the formation of more structured, hierarchical candidate solutions. (Koza's (1992, 1994) "automatic definition of functions" is one way in which GP can be encouraged to design hierarchically structured programs.) Also, the resulting trees, being large, can be very difficult to understand and to simplify. Systematic experiments evaluating the usefulness of tree encodings and comparing them with other encodings are only just beginning in the genetic programming community. These are only the most common encodings; a survey of the GA literature will turn up experiments on several others.

How is one to decide on the correct encoding for one's problem? Lawrence Davis, a researcher with much experience applying GAs to real world problems, strongly advocates using whatever encoding is the most natural for your problem, and then devising a GA that can use that encoding (Davis 1991). Until the theory of GAs and encodings is better formulated, this might be the best philosophy; most research is currently done by guessing at an appropriate encoding and then trying out a particular version of the GA on it. This is not much different from other areas of machine learning; for example, encoding a learning problem for a neural net is typically done by trial and error. One appealing idea is to have the encoding itself adapt so that the GA can make better use of it. Likewise, tree encodings such as those used in genetic programming automatically allow for adaptation of the encoding, since under crossover and mutation the trees can grow or shrink. Meyer and Packard's encoding of condition sets also allowed for individuals of varying lengths, since crossovers between individuals of different lengths could cause the number of conditions in a set to increase or decrease.

E. Octal Encoding

This encoding uses string made up of octal numbers (0-7)

Chromosome 1	13578327
Chromosome 2	26834425

Fig. 3.4 Octal Encoding

F. The Fitness Function

The operation of fitness proportionate reproduction for the genetic programming paradigm is the basic engine of Darwinian reproduction and survival of the fittest. Each individual in a population is assigned a fitness value as a result of its interaction with the environment. The fitness function is an equation that is a function of including properties (Genes) in each string (Chromosome). The general form of this function depends to the studying problem and mentions the final goal of problem. Fitness is the driving force of Darwinian natural selection and, likewise, of genetic algorithms. Note that the parents remain in the population while this operation is performed and therefore can potentially participate repeatedly in this operation (and other operations) during the current generation. That is, the selection of parents is done with replacement (i.e. reselection) allowed. For example, in a optimization problem this function can be the objective function and then the maximize or minimize of objective can be the final goal of problem. Having decoded the chromosome representation into the decision variable domain, it is possible to assess the performance, or *fitness*, of individual members of a population. This is done through an objective function that characterizes an individual's performance in the problem domain. In the natural world, this would be an individual's ability to survive in its present environment. Thus, the objective function establishes the basis for selection of pairs of individuals that will be mated together during reproduction. During the reproduction phase, each individual is assigned a fitness value derived from its raw performance measure given by the objective function. This value is used in the selection to bias towards more fit individuals. Highly fit individuals, relative to the whole population, have a high probability of being selected for mating whereas less fit individuals have a correspondingly low probability of being selected. Once the individuals have been assigned a fitness value, they can be chosen from the population, with a probability according to their relative fitness, and recombine to produce the next generation.

The objective function is used to provide a measure of how individuals have performed in the problem domain. In the case of a minimization problem, the most fit individuals will have the lowest numerical value of the associated objective function. This raw measure of fitness is usually only used as an intermediate stage in determining the relative performance of individuals in a GA. Another function, the *fitness function*, is normally used to transform the objective function value into a measure of relative fitness thus: where f is the objective function, g transforms the value of the objective function to a non-negative number and F is the resulting relative fitness. This mapping is always necessary when the objective function is to be minimized as the lower objective function values correspond to fitter individuals. In many cases, the fitness function value corresponds to the number of offspring that an individual can expect to produce in the next generation. A commonly used transformation is that of proportional fitness assignment .

G. Loss Function

The loss function is an auxiliary function that his objective is the incorporation of loss resulting from constraints violation into objective function. Generally the constraints are divided into the two groups: 1- Hard constraints, 2- Soft constraints. The hard constraint concludes some constraints that must be observed during the process. The soft constraints is the other type of limitation that may be observed or not, but the former should results some losses or costs. The design of loss function is very important in many genetic algorithm problems and decreases the convergence time. It is also possible for the fitness function to consider secondary factors (e.g., efficiency of the S-expression, parsimony of the S-expression, compliance with the initial conditions of a differential equation, etc.). When the number of environmental cases is large, it is sometimes expeditious to compute the fitness function using only a sampling of the possible environmental cases (including, possibly, a sampling that varies from generation to generation to minimize the possible bias resulting from such sampling).

H. Reproduction

Reproduction is an operator that makes more copies of better strings in a new population. Reproduction is usually the first operator applied on a population. Reproduction selects good strings in a population and forms a mating pool.

Thus, in reproduction operation the process of natural selection causes those individuals that encode successful structures to produce copies more frequently. To sustain the generation of a new population, the reproduction of the individuals in the current population is necessary. For better individuals, these should be from the fittest individuals of the previous population. GAs mimics the survival-of-the-fittest principle of nature to make a search process. In genetic algorithm, fitness is used to allocate reproductive traits to the individuals in the population and thus act as some measure of goodness to be maximized. This means that individuals with higher fitness value will have higher probability of being selected as candidates for further examination. At this stage individuals are selected randomly from the population in order of the fit individual. This shows that individuals with poor fitness status are removed from the next population. Therefore this stage of the GA helps to choose good and more fitted individuals from the population to recombine, producing new and well fitted offspring for the next generation. Certain GA operators are then used on the selected candidates to recombine their chromosomes. There exist a number of reproduction operators in GA literature, but the essential idea in all of them is that the above average strings are picked from the current population and their multiple copies are inserted in the mating pool in a probabilistic manner. Fitness in biological sense is a quality value which is a measure of the reproductive efficiency of chromosomes.

III. OPERATORS IN GENETIC ALGORYTHM (GA)

In this section we describe some of the selection, crossover, and mutation operators commonly used in genetic algorithms.

A. Selection Processes in Genetic Algorithm (GA)

Selection is usually the first operator applied on a population after encoding. Selection is an operator that makes more copies of better strings in a new population. Selection is the process of determining the number of times, or *trials*, a particular individual is chosen for reproduction and, thus, the number of offspring that an individual will produce. It is the component which guides the algorithm to the solution by preferring individuals with high fitness over low-fitted ones. It can be a deterministic operation, but in most implementations it has random components. The purpose of selection is, of course, to emphasize the fitter individuals in the population in hopes that their offspring will in turn have even higher fitness. In selection process a chromosome is selected according to its objective function measurement (Biologist called it fitness function). This function can show some specific measurement such as utility, benefit or any other objectives that should be maximized or minimized. The useful chromosome to copy is determined according to the same functions. The selection function identifies promising genetic material and determines how much of it should be present in the next generation. The genetic material that composes genotypes of higher quality tend to have a higher probability of finding itself reused in some form or other in the next generation. In most GP systems, selection is applied at the organism level and the term "mating pool" is taken to mean the storage containing the individuals selected for reproduction. While GP systems typically select genotypes, there are other possibilities. In particular, selecting whole sets of genotypes (such as schemata) or combinations of sets and individual genotypes is also possible.

A *genotype selection scheme* determines the probability that a genotype will be selected for producing offspring by crossover or mutation. In order to search for increasingly fitter phenotypes, higher selection probabilities are assigned to the genotypes of better scoring phenotypes. The selection operator selects genotypes based on the general principle that the fitter the individual, the higher its probability of being selected for reproduction should be. Many selection methods have been proposed, examined and compared. Just as the case for encodings, these descriptions do not provide rigorous guidelines for which method should be used for which problem; this is still an open question for GAs. (For more technical comparisons of different selection methods, see Goldberg and Deb 1991, Bäck and Hoffmeister 1991, and Hancock 1994.)

Methods are as follows;

- Roulette Wheel Selection (Fitness-Proportionate Selection)
- Tournament selection
- Boltzman selection
- Rank selection
- Linear ranking selection
- Exponential ranking selection
- Steady state selection
- Elitism selection

- Truncation selection

Selection might operate with or without replacement. With replacement, the solution that is added to the new parent population is kept in the combined population and a good solution can be chosen more than once for the new parent population. The term with replacement is used since the original genotype is available for further selection as additional genetic material is selected. The system maintains no memory of prior selection. Without replacement, the selected solution is placed in the parent population and removed from the combined population and each solution can only be selected once for the new parent population. Selection provides the driving force in genetic algorithms. With too much force, genetic search will terminate prematurely. In this way, the selection directs the genetic search toward promising regions in the search space and that will improve the performance of genetic algorithms

B. Roulette Wheel Selection (Fitness-Proportionate Selection)

Holland's original GA used fitness−proportionate selection, in which the "expected value" of an individual (i.e., the expected number of times an individual will be selected to reproduce) is that individual's fitness divided by the average fitness of the population. The most common method for implementing this is "roulette wheel" sampling. The roulette wheel selection is the simplest method that selects the best chromosome according to the ratio of each chromosomes fitness to sum of all fitness values related to all chromosomes (roulette-wheel selection (Holland, 1975; Goldberg, 1989). Roulette wheel selection is most common selection method used in genetic algorithms for selecting potentially useful individuals (solutions) for crossover and mutation. In roulette wheel selection, as in all selection methods, possible solutions are assigned a fitness by the fitness function. This fitness level is used to associate a probability of selection with each individual. While candidate solutions with a higher fitness will be less likely to be eliminated, there is still a chance that they may be. We can force the property to be satisfied by applying a random experiment which is, in some sense, a generalized roulette game. In this roulette game, the slots are not equally wide, i.e. the different outcomes can occur with different probabilities Roulette wheel is a specific game. At this game a roulette wheel is rolled around a central point and then a specific area is selected when it stops. To produce a simple roulette wheel, the ratio of each string fitness to the sum of all fitness values in population is calculated. This ratio is determined as an area of roulette wheel. Each individual is assigned a slice of a circular "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 can be implemented as follows:

1. Sum the total expected value of individuals in the population. Call this sum *T*.
2. Choose a random integer *r* between 0 and *T*.
3. 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.

OR

The roulette wheel selection scheme can be implemented as follows: In this method all the chromosomes (individuals) in the population are placed on the roulette wheel according to their fitness value. Each individual is assigned a segment of roulette wheel. The size of each segment in the roulette wheel is proportional to the value of the fitness of the individual - the bigger the value is, the larger the segment is. Then, the virtual roulette wheel is spinned. The individual corresponding to the segment on which roulette wheel stops are then selected. The process is repeated until the desired number of individuals is selected. Individuals with higher fitness have more probability of selection. This may lead to biased selection towards high fitness individuals. It can also possibly miss the best individuals of a population. There is no guarantee that good individuals will find their way into next generation. Roulette wheel selection uses exploitation technique in its approach. The average fitness of the population for i^{th} generation in roulette wheel selection is calculated as the average fitness of the population for i^{th} generation in roulette wheel selection is calculated as

$$\overline{FRW_{ij}} = \frac{\sum_{j=1}^{N} FRW_J}{N}$$

where i varies from 1 to ngen and j varies from 1 to N.

Therefore, the probability for selecting the j^{th} string is

$$PRW_j = \frac{FRW}{\sum_{j=1}^{N} FRW}$$

Where;

ngen → total number of generations

N → total population size

$FRWi, j$ → fitness of j^{th} individual in i^{th} generation for roulette wheel selection

FRW_j → Average Fitness of the population in j^{th} generation in Roulette Wheel Selection

Parents are selected according to their fitness i.e., each individual is selected with a probability proportional to its fitness value. In other words, depending on the percentage contribution to the total population fitness, string is selected for mating to form the next generation. This way, weak solutions are eliminated and strong solutions survive to form the next generation. With roulette wheel selection there is a chance some weaker solutions may survive the selection process; this is an advantage, as though a solution may be weak, it may include some component which could prove useful following the recombination process. In roulette-wheel selection, each individual in the population is assigned a roulette wheel slot sized in proportion to its fitness. That is, in the biased roulette wheel, good solutions have a larger slot size than the less fit solutions. The roulette wheel is spun to obtain a reproduction candidate.

To illustrate, For example, consider a population containing four strings shown in the table below. Each string is formed by concatenating four substrings which represents variables a,b,c and d. Length of each string is taken as four bits. The first column represents the possible solution in binary form. The second column gives the fitness values of the decoded strings. The third column gives the percentage contribution of each string to the total fitness of the population. Then by "Roulette Wheel" method, the probability of candidate 1 being selected as a parent of the next generation is 28.09%. Similarly, the probability that the candidates 2, 3, 4 will be chosen for the next generation are 19.59, 12.89 and 39.43 respectively. These probabilities are represented on a pie chart, and then four numbers are randomly generated between 1 and 100. Then, the likeliness that the numbers generated would fall in the region of candidate 2 might be once, whereas for candidate 4 it might be twice and candidate 1 more than once and for candidate 3 it may not fall at all. Thus, the strings are chosen to form the parents of the next generation.

Table 3.1

D Nagesh Kumar, IISc, Bangalore M8L5 Optimization Methods: Advanced Topics in Optimization - Evolutionary Algorithms for Optimization and Search 5 **Candidate**	**Fitness value**	**Percentage of total fitness**
1011 0110 1101 1001	109	28.09
0101 0011 1110 1101	76	19.59
0001 0001 1111 1011	50	12.89
1011 1111 1011 1100	153	39.43
Total	388	100

Many selection techniques employ a “roulette wheel” mechanism to probabilistically select individuals based on some measure of their performance. Stochastic universal sampling (SUS) is a single-phase sampling algorithm with minimum spread and zero bias. Instead of the single selection pointer employed in roulette wheel methods, SUS uses *N* equally spaced pointers, where *N* is the number of selections required. This stochastic method statistically results in the expected number of offspring for each individual. However, with the relatively small populations typically used in GAs, the actual number of offspring allocated to an individual is often far from its expected value (an extremely unlikely series of spins of the roulette wheel could even allocate all offspring to the worst individual in the population). Rather than spin the roulette wheel *N* times to select *N* parents, SUS spins the wheel once—but with *N* equally spaced pointers, which are used to selected the *N* parents. The population is shuffled randomly and a single random number in the range [0 *Sum*/*N*] is generated, *ptr*. The *N* individuals are then chosen by generating the *N* pointers spaced by 1, [*ptr*, *ptr*+1, ..., *ptr*+*N*-1], and selecting the individuals whose fitnesses span the positions of the pointers. An individual is thus guaranteed to be selected a minimum of times and no more than , thus achieving minimum spread. In addition, as individuals are selected entirely on their position in the population, SUS has zero bias. The roulette wheel selection methods can all be implemented as O (NlogN) although SUS is a simpler algorithm and has time complexity O(N).

SUS does not solve the major problems with fitness−proportionate selection. Typically, early in the search the fitness variance in the population is high and a small number of individuals are much fitter than the others. Under fitness−proportionate selection, they and their descendents will multiply quickly in the population, in effect preventing the GA from doing any further exploration. This is known as "premature convergence." In other words, fitness−proportionate selection early on often puts too much emphasis on "exploitation" of highly fit strings at the expense of exploration of other regions of the search space. Later in the search, when all individuals in the population are very similar (the fitness variance is low), there are no real fitness differences for selection to exploit, and evolution grinds to a near halt. Thus, the rate of evolution depends on the variance of fitness in the population

C. Tournament Selection Method

The other alternative to strict fitness-proportional selection is *tournament selection* (Goldberg et al., 1989) in which a set of chromosomes is chosen and compared, the best one being selected for Parenthood. In this method a group of individuals are chosen at random form and the individual with the highest fitness is selected for inclusion in the next generation. Selection pressure can be easily adjusted by changing the tournament size. If the tournament size is larger, weaker individuals are less likely to be selected. This process is repeated until the appropriate numbers of individuals are selected for the new generation. In tournament selection a string is only selected when it successes to other competitors or on the other hand it's fitness is highest than the other competitors. If sampling is down without replacing, then the probability selection of a string with lower fitness relative to string I is equal to the probability of a selected uniform stochastic number is less than or equal to: [1]The number of individual in the set is called the tournament size. In tournament selection, *s* chromosomes are chosen at random (either with or without replacement) and entered into a tournament against each other. The fittest individual in the group of *k* chromosomes wins the tournament and is selected as the parent. The most widely used value of *s* is 2. Using this selection scheme, *n* tournaments are required to choose *n* individuals. That is a common tournament size is 2, this is called binary tournament. By adjusting tournament size, the selection pressure can be made arbitrarily large or small. For example, using large Tournament size has the effect of increasing the selection pressure, since below average individuals are less likely to win a tournament while above average individuals are more likely to win it. Two individuals are chosen at random from the population. A random number *r* is then chosen between 0 and 1. If $r < k$ (where *k* is a parameter, for example 0.75), the fitter of the two individuals is selected to be a parent; otherwise the less fit individual is selected. The two are then returned to the original population and can be selected again. An analysis of this method was presented by Goldberg and Deb (1991).
One potential advantage of tournament selection over all other forms is that it only needs a preference ordering between pairs or groups of strings, and it can thus cope with situations where there is no formal objective function at all in other words, it can deal with a purely *subjective* objective function! However, we should point out again that tournament selection is also subject to arbitrary stochastic effects in the same way as roulette-wheel selection there is no guarantee that every string will appear in a given cycle. Indeed, using sampling with replacement there is a probability of approximately that a given string will not appear at all. One way of coping with this, at the expense of a little extra computation, is to use a variance reduction technique from simulation theory.

D. Boltzmann Selection

Boltzmann selection is a method inspired by the technique of simulated annealing. In Boltzman selection, selection pressure is slowly increased over time to gradually focus the search. The inspiration comes from annealing in metallurgy, a technique involving heating and controlled cooling of a material to increase the size of its crystals and

reduce their defects. By analogy with this physical process, each step of this selection algorithm replaces a current individual by a random "nearby" solution, chosen with a probability that depends on the difference between the corresponding function values and on a global parameter *T* called the temperature. The temperature is gradually decreased during the process, creating a dependency such that the current solution changes almost randomly when *T* is large, but increasingly "downhill" as *T* goes to zero. The allowance for "uphill" moves saves the method from becoming stuck at local minima which are the bane of greedier methods. A typical implementation is to assign to each individual *i* an expected value, where *T* is temperature and <>*t* denotes the average over the population at time *t*. Experimenting with this formula will show that, as *T* decreases, the difference in ExpVal(*i*,*t*) between high and low fitnesses increases. The desire is to have this happen gradually over the course of the search, so temperature is gradually decreased according to a predefined schedule.
The selection probability is as follows for this method:

$$P_i = \frac{\exp\left(Bf_i\right)}{Z}$$

Where B control the selection intensity and

$$Z = \sum_{j=1}^{n} \exp\left(bf_j\right)$$

Rogers and Prugel-Bennett proposed the selection intensity is nearly determined using B. Fitness proportionate selection is commonly used in GAs mainly because it was part of Holland's original proposal and because it is used in the Schema theorem, but, evidently, for many applications simple fitness‒proportionate selection requires several "fixes" to make it work well.

Proposed Annealed Selection
The proposed selection approach is to move the selection criteria from exploration to exploitation so as to obtain the perfect blend of the two techniques. In this method, fitness value of each individual is computed. Depending upon the current generation number of genetic algorithm, selection pressure is changed and new fitness contribution, $X_{i,j}$ of each individual is computed. Selection probability of each individual is computed on the basis of $X_{i,j}$. As the generation of population changes, fitness contribution changes and selection probability of each individual also changes. The proposed blended selection operator computes fitness of individual depending on the current number of generation as under:

$$FX_i = \frac{FRW_i}{(ngen+1) - nogen}$$

The probability for selecting the i^{th} string is

$$PX_i = \frac{FX_i}{\sum_{i=1}^{N} FX_i}$$

Where;
FX_i → Average Fitness of the population in i^{th} generation in Proposed Blended Selection
FRW_i → Average Fitness of the population in i^{th} generation in Roulette Wheel Selection
ngen → total number of generations
nogen → current number of generation

E. Rank Selection
In ranking selection, the individuals in the population are sorted from best to worst according to their fitness values. Each individual in the population is assigned a numerical rank based on fitness, and selection is based on this ranking rather than differences in fitness. *Ranking selection* sorts the genotypes of a generation according to the raw fitness score of their associated genotypes. The probability of a genotype being selected for reproduction depends only on its position in terms of fitness relative to the other genotypes and not on the actual fitness score. The previous selection will have problems when the fitness differs very much. For example, if the best chromosome fitness is 90% of all the roulette wheel then the other chromosomes will have very few chances to be selected. Rank selection first ranks the

population and then every chromosome receives fitness from this ranking. The worst will have fitness 1, second worst 2 etc. and the best will have fitness N (number of chromosomes in population). After this all the chromosomes have a chance to be selected. But this method can lead to slower convergence, because the best chromosomes do not differ so much from other ones.

Rank Selection sorts the population first according to fitness value and ranks them. Then every chromosome is allocated selection probability with respect to its rank. Individuals are selected as per their selection probability. Rank selection is an explorative technique of selection. Rank selection prevents too quick convergence and differs from roulette wheel selection in terms of selection pressure. Rank selection overcomes the scaling problems like stagnation or premature convergence when the selection has caused the search to narrow down too quickly. Ranking controls selective pressure by uniform method of scaling across the population. Rank selection behaves in a more robust manner than other methods. In Rank Selection, sum of ranks is computed and then selection probability of each individual is computed as under:

$$rsum_i = \sum_{i=1}^{N} r_{i,j}$$

where i varies from 1 to ngen and j varies from 1 to N.

$$PRANK_i = \frac{r_{i,j}}{rsum_i}$$

Where;

$rsum_i$ $\rightarrow$ sum of ranks in i^{th} generation

$r_{i,j}{}^{th}$ $\rightarrow$ rank of j^{th} individual in i^{th} generation for rank selection

ngen $\rightarrow$ total number of generations

The advantage of this method is that it can prevent very fit individuals from gaining dominance early at the expense of less fit ones, which would reduce the population's genetic diversity and might hinder attempts to find an acceptable solution. The disadvantage of this method is that it required sorting the entire population by rank which is a potentially time consuming procedure.

F. Linear Ranking Selection

In linear ranking selection method, the strings are ranked according to their fitness and the selection probability is corresponding to the form of ranking in a string. It can be determined as follows:

$$P_i = \frac{\left(n^+ + \frac{n^+ - n^-}{n-1}(i-1)\right)}{Z}$$

Where (n^+) and (n^-) are the proper number of copies from the best and the worth chromosomes, respectively. n=Z is a normalizer parameter.

The sum of $n^+ + n^-$ should be equal 2 to make sure probability is less than 1.

G. Exponential Ranking Selection

An alternative to the week linear ranking is to assign survival probabilities to the sorted individuals using an exponential function:

$$P_i = \frac{C^{n-i}}{Z}$$

where *C∈[0,1]* is a parameter of algorithm. Therefore

$$Z = \sum_{j=1}^{n} C^{n-j} = \frac{C-1}{C^n - 1}$$

Where P_i can be simplify as follows:

$$P_i = \frac{C-1}{C^n - 1} C^{n-i}$$

According to these Figures the exponential ranking has more intensity selection relative to linear method. However some C values results high selection intensity, exponential ranking also results in a distribution with low variance. A steep loss in variance is also problematic because it suggests that the population loses diversity very rapidly making the algorithm more likely to coverage to a suboptimal solution unless the variation operators reintroduce sufficient diversity.

H. Steady State Selection

The steady state selection will eliminate the worst of individuals in each generation. This is not a particular method of selecting parents. The main idea of this selection is that big part of chromosomes should survive to the next generation. In steady−state selection, only a few individuals are replaced in each generation: usually a small number of the least fit individuals are replaced by offspring resulting from crossover and mutation of the fittest individuals. Steady−state GAs are often used in evolving rule−based systems (e.g., classifier systems; see Holland 1986) in which incremental learning (and remembering what has already been learned) is important and in which members of the population collectively (rather than individually) solve the problem at hand. GA then works in the following way. In every generations a few (good — with high fitness) chromosomes are selected for creating a new offspring. Then some (bad —with low fitness) chromosomes are removed and the new offspring is placed in their place. The rest of population survives to the new generation.

I. Elitism Selection

"Elitism," first introduced by Kenneth De Jong (1975), is an addition to many selection methods that forces the GA to retain some number of the best individuals at each generation. It improves the selection process and save the best individuals. With elitist selection, the quality of the best solution in each generation monotonically increases over time. Without elitist selection, it is possible to lose the best individuals due to stochastic errors (due to crossover, mutation or selection pressure). The ratio *N:M* between the elite size, *N*, and the population size, *M*, is called the *elite fraction*. Elitism is commonly used in generational GP to ensure that the best individuals discovered in a generation are not lost, and are made available for possible further improvements to new generations. As selection is probabilistic, such individuals might otherwise be lost if they're not selected to reproduce. If the selection method guarantees the conservation of some individuals, it is called *elitist*. Such individuals can be lost if they are not selected to reproduce or if they are destroyed by crossover or mutation. Many researchers have found that elitism significantly improves the GA's performance. The idea of elitism has already been introduced. When creating a new population by crossover and mutation, we have a big chance that we will loose the best chromosome. Elitism is the name of a method which first copies the best chromosome (or a few best chromosomes) to the new population. The rest is done in the classical way. Elitism can very rapidly increase performance of GA because it prevents losing the best found solution.

J. Sigma Scaling Selection

To address such problems, GA researchers have experimented with several "scaling" methods—methods for mapping "raw" fitness values to expected values so as to make the GA less susceptible to premature convergence. One example is "sigma scaling" (it was called "sigma truncation" in Goldberg 1989a), which keeps the selection pressure (i.e., the degree to which highly fit individuals are allowed many offspring) relatively constant over the course of the run rather than depending on the fitness variances in the population. Under sigma scaling, an individual's expected value is a function of its fitness, the population mean, and the population standard deviation. A example of sigma scaling would be where ExpVal(*i*,*t*) is the expected value of individual *i* at time *t*, *f(i)* is the fitness of *i*, *f(t)* is the mean fitness of the population at time *t*, and Ã(*t*) is the standard deviation of the population fitness at time *t*. This function, used in the work of Tanese (1989), gives an individual with fitness one standard deviation above the mean 1.5 expected offspring. If ExpVal(*i*,*t*) was less than 0, Tanese arbitrarily reset it to 0.1, so that individuals with very low fitness had some small chance of reproducing. At the beginning of a run, when the standard deviation of fitness is typically high, the fitter individuals will not be many standard deviations above the mean, and so they will not be allocated the lion's share of offspring. Likewise, later in the run, when the population is typically more converged and the standard deviation is typically lower, the fitter individuals will stand out more, allowing evolution to continue.

K. Truncation Selection

In *truncation selection* (Muhlenbein and Schlierkamp-Voosen, 1993) individuals are sorted according to their fitness. Only the best individuals are selected as parents. The parameter for truncation selection is the *truncation threshold*. It indicates the proportion of the population to be selected as parents and takes values ranging from 50%-10%. Individuals below the truncation threshold do not produce offspring. Unlike fitness proportionate selection there are no chances for weaker solutions to survive the selection process. This method deterministically selects the best strings in population. This deterministic method can be discussed in a probabilistic framework as follows:

$$P_i = \begin{cases} 0 & if \leq (n-\tau) \\ \dfrac{1}{\tau} & i \geq (n-\tau) \end{cases}$$

This shows the mean and variance of selected strings for different τ. Selection a small value for τ increases the mean of population and also decreases the variance of population, rapidly.

IV. PARAMETERS FOR GENETIC ALGORITHMS

The operation of GAs begins with a population of a random string representing design or decision variables. The population is then operated by three main operators; reproduction, crossover and mutation to create a new population of points. GAs can be viewed as trying to maximize the fitness function, by evaluating several solution vectors. The purpose of these operators is to create new solution vectors by selection, combination or alteration of the current solution vectors that have shown to be good temporary solutions. The new population is further evaluated and tested till termination. If the termination criterion is not met, the population is iteratively operated by the above three operators and evaluated. This procedure is continued until the termination criterion is met. One cycle of these operations and the subsequent evaluation procedure is known as a generation in GAs terminology. The operators are described below.

V. CROSSOVER

In most GAs, individuals are represented by fixed-length strings and crossover operates on pairs of individuals (parents) to produce new strings (offspring) by exchanging segments from the parents' strings. Traditionally, the number of crossover points (which determines how many segments are exchanged) has been fixed at a very low constant value of 1 or 2. Support for this decision came from early work of both a theoretical and empirical nature by Holland. In spite of this, other GAs problems were implemented using other types of crossover. The crossover (recombination) operation for the genetic programming paradigm creates variation in the population by producing offspring that combine traits from two parents. Crossover is simply a matter of replacing some of the genes in one parent by the corresponding genes of the other. In sexual reproduction, as it appears in the real world, the genetic material of the two parents is mixed when the gametes of the parents merge. Usually, chromosomes are randomly split and merged, with the consequence that some genes of a child come from one parent while others come from the other parents. This mechanism is called crossover. One of the unique aspects of the work involving genetic algorithms (GAs) is the important role that Crossover plays in the design and implementation of robust evolutionary systems. A crossover operator is used to recombine two strings to get a better string. In crossover operation, recombination process creates different individuals in the successive generations by combining material from two individuals of the previous generation. . Crossover is a very powerful tool for introducing new genetic material and maintaining genetic diversity, but with the outstanding property that good parents also produce well-performing children or even better ones. Several investigations have come to the conclusion that crossover is the reason why sexually reproducing species have adapted faster than asexually reproducing ones. Basically, crossover is the exchange of genes between the chromosomes of the two parents. In the simplest case, we can realize this process by cutting two strings at a randomly chosen position and swapping the two tails.

In reproduction, good strings in a population are probabilistic-ally assigned a larger number of copies and a mating pool is formed. It is important to note that no new strings are formed in the reproduction phase. In the crossover operator, new strings are created by exchanging information among strings of the mating pool. The two strings participating in the crossover operation are known as parent strings and the resulting strings are known as children strings. It is intuitive from this construction that good sub-strings from parent strings can be combined to form a better child string, if an appropriate site is chosen. With a random site, the children strings produced may or may not have a combination of good sub-strings from parent strings, depending on whether or not the crossing site falls in the appropriate place. But this is not a matter of serious concern, because if good strings are created by crossover, there will be more copies of them in the next mating pool generated by crossover. It is clear from this discussion that the effect of cross over may be detrimental or beneficial. Thus, in order to preserve some of the good strings that are already present in the mating pool, all strings in the mating pool are not used in crossover.

A crossover operator is mainly responsible for the search of new strings even though mutation operator is also used for this purpose sparingly. Crossover rate determines the probability that crossover will occur. The crossover will generate new individuals in the population by combining parts of existing individuals. The crossover rate is usually high and 'application dependent'. Many researchers suggest crossover rate to be between 0.6 and 0.95. The underlying objective of crossover is to exchange information between strings to get a string that is possibly better than the parents. After selection process, the basic operator for producing new chromosomes which is the crossover operator, exchanges the main part of each two selected chromosome to save the best properties of these two strings. In this Section, a number of variations on crossover are described (Goldberg, 1989; Spears, 1997) and discussed and the relative merits of each reviewed.

- One-Point Crossover (Single-Point Crossover)
- Two-Point Crossover
- Multi Point Crossover (N Point Crossover)
- Uniform Crossover
- Three Parent Crossover
- Segmented Crossover
- Shuffle Crossover
- Uniform Order-Based Crossover
- Order-Based Crossover
- Cycle Crossover (CX)
- Evolving Crossover "Hot Spots"
- Non-linear Crossover
- Reorder Crossover
- Arithmetic Crossover
- Heuristic Crossover
- Crossover with Reduced Surrogate
- Partially Matched Crossover (PMX)
- Precedence Preservative Crossover (PPX)
- Crossover Probability

A. One-Point Crossover (Single Point Crossover)

A commonly used method for crossover is called single point crossover. In this method, a single point crossover position (called cut point) is chosen at random (e.g., between the 4th and 5th variables) and the parts of two parents after the crossover position are exchanged or swapped to form two offspring. In one-point crossover, a crossover site is selected at random over the string length, and the alleles on one side of the site are exchanged between the individuals In one site crossover, a crossover site is selected randomly (shown as vertical lines). The portion right of the selected site of these two strings is exchanged to form a new pair of strings. The new strings are thus a combination of the old strings. One site crossover is more suitable when string length is small. Really, the genetic algorithm attempt to make a new string with crossover that maintains all good properties of two crossed strings. The number of strings that their parts are exchanged is controlled with cross over probability. The crossing over between two selected chromosomes (strings) is down with a specific probability that changes from 0.5 to 1. (i.e., selected chromosomes have this probability of being used in crossover). In one simple crossover, one cross point is determined stochastically along to the chromosome.

In one point crossover, selected pair of strings is cut at some random position and their segments are swapped (crossover) to form new pair of strings. Then according to one-point crossover, if a random crossover point is chosen from left and segments are cut as shown below:

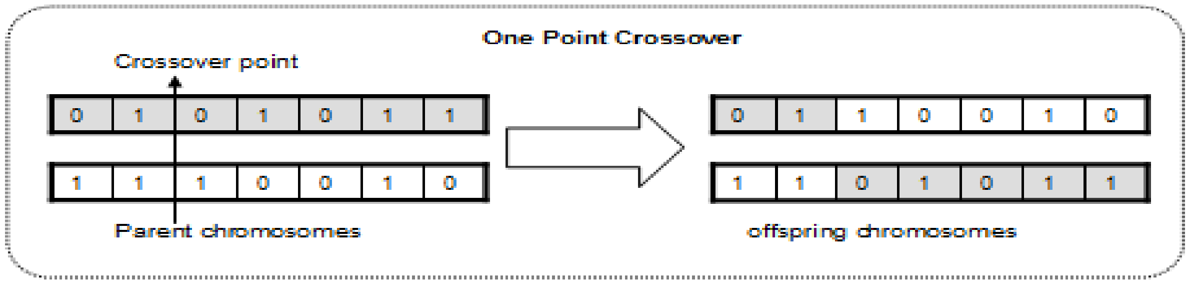

Fig. 3.5 One-point crossover operation

B. Two-Point Crossover

To reduce bias and endpoint effect, many GA practitioners use two-point crossover, in which two positions are chosen at random and the segments between them are exchanged. Two-point crossover is less to disrupt schemas with large likely defining lengths and can combine more schemas than single-point crossover. In addition, the segments that are exchanged do not necessarily contain the endpoints of the strings. Again, there are schemas that two-point crossover cannot combine. As mentioned, the two-point crossover operator randomly selects points within chromosome then interchanges the two parent chromosomes between these two points to produce two new offspring for mating in the next generation.

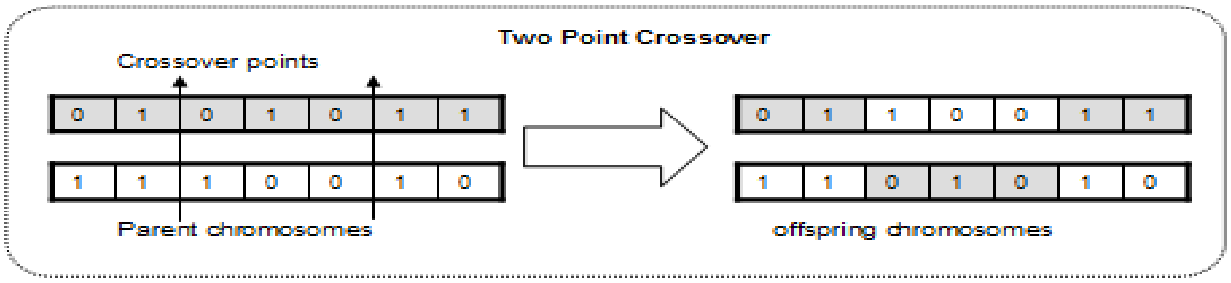

Fig. 3.6 Two-point crossover

In figure 3.6, the arrows 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.

B. Multi Point Crossover (N Point Crossover)

Multi-point crossover is a generalization of single point crossover, introducing a higher number of cut-points. In this case multi positions are chosen at random and the segments between them are exchanged. Instead of only one, N breaking points are chosen randomly. Every second section is swapped. Among this class, two-point crossover is particularly important. For multi-point crossover, *n* crossover positions, , where *ki* are the crossover points and *l* is the length of the chromosome, are chosen at random with no duplicates and sorted into ascending order. Then, the bits between successive crossover points are exchanged between the two parents to produce two new offspring. The section between the first allele position and the first crossover point is not exchanged between individuals. The idea behind multi-point, and indeed many of the variations on the crossover operator, is that the parts of the chromosome representation that contribute to the most to the performance of a particular individual may not necessarily be contained in adjacent substrings. Further, the disruptive nature of multi-point crossover appears to encourage the exploration of the search space, rather than favoring the convergence to highly fit individuals early in the search, thus making the search more robust.

In two-point scheme, there will be two break points in the strings that are randomly chosen. At the break-point, the segments of the two strings are swapped so that new set of strings are formed.

C. Uniform Crossover

Another common recombination operator is uniform crossover (Syswerda, 1989; Spears and De Jong, 1994). The uniform crossover is a more general method. In uniform crossover, every allele is exchanged between the pair of randomly selected chromosomes with a certain probability, *pe*, known as the swapping probability. Usually the swapping probability value is taken to be 0.5. Uniform crossover does not use cut-points, but simply uses a global parameter to indicate the likelihood that each variable should be exchanged between two parents. Uniform crossover, like multi-point crossover, has been claimed to reduce the bias associated with the length of the binary representation used and the particular coding for a given parameter set. This helps to overcome the bias in single-point crossover towards short substrings without requiring precise understanding of the significance of individual bits in the chromosome representation. Spears and De Jong, 1994 have demonstrated how uniform crossover may be parameterized by applying a probability to the swapping of bits. This extra parameter can be used to control the amount of disruption during recombination without introducing a bias towards the length of the representation used. When uniform crossover is used with real-valued alleles, it is usually referred to as *discrete recombination*. In this method some independent genes (bits) is selected from each string stochastically and then are exchanged. However, for a uniform crossover following steps should be proceeded (Gen and Cheng, 2000):

1. Two chromosomes is selected by selection operator.
2. A part of parent chromosome is stochastically used to form a part of childes.
3. The second step is repeated until the total part of child is completed from.

Consider the following two parents, crossover mask and resulting offspring:

Parent 1	1 0 1 1 0 0 0 1 1 1
Parent 2	0 0 0 1 1 1 1 0 0 0
Mask	0 0 1 1 0 0 1 1 0 0
Offspring 1	0 0 1 1 1 1 0 1 0 0
Offspring 2	1 0 0 1 0 0 1 0 1 1

Fig 3.7 Uniform crossover

Here, the first offspring, 1, is produced by taking the bit from 1 if the corresponding mask bit is 1 or the bit from 2 if the corresponding mask bit is 0. Offspring 2 is created using the inverse of the mask or, equivalently, swapping P1 and P2.

D. 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.

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
Offspring	0 1 1 0 1 0 0 1

Fig 3.8 Three Parent Crossover

E. Segmented crossover

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

F. Shuffle crossover

Another related crossover operator is that of *shuffle* First a randomly chosen permutation is applied to the two parents, then N-point crossover is applied to the shuffled parents, finally, the shuffled children are transformed back with the inverse permutation. Thus a single cross-point is selected, but before the bits are exchanged, they are randomly shuffled in both parents. After recombination, the bits in the offspring are unshuffled. This too removes positional bias as the bits are randomly reassigned each time crossover is performed.

G. Ordered Crossover

Ordered two-point crossover is used when the problem is of order based, for example in U-shaped assembly line balancing etc. Given two parent chromosomes, two random crossover points are selected partitioning them into a left, middle and right portion. 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 illustrated below

Parent 1	4 2 | 1 3 | 6 5
Parent 2	2 3 | 1 4 | 5 6
Offspring 1	4 2 | 3 1 | 6 5
Offspring 2	2 3 | 4 1 | 5 6

Fig 3.9 Ordered Crossover

H. Uniform Order-Based Crossover

The *n*-point and uniform crossover methods described above are not well suited for search problems with permutation codes such as the ones used in the traveling salesman problem. They often create offspring that represent invalid solutions for the search problem. Therefore, when solving search problems with permutation codes, a problem-specific repair mechanism is often required (and used) in conjunction with the above recombination methods to always create valid candidate solutions. In uniform order-based crossover, two parents (say P1 and P2) are randomly selected and a random binary template is generated. Some of the genes for offspring C1 are filled by taking the genes from parent P1 where there is a one in the template. At this point we have C1 partially filled, but it has some "gaps". The genes of parent P1 in the positions corresponding to zeros in the template are taken and sorted in the same order as they appear in parent P2. The sorted list is used to fill the gaps in C1. Offspring C2 is created by using a similar process.

I. Order-Based Crossover

The order-based crossover operator (Davis, 1985) is a variation of the uniform order-based crossover in which two parents are randomly selected and two random crossover sites are generated. The genes between the cut points are copied to the children. Starting from the second crossover site copy the genes that are not already present in the offspring from the alternative parent (the parent other than the one whose genes are copied by the offspring in the initial phase) in the order they appear. For example, as shown in Figure 4.3, for offspring C1, since alleles C, D, and E are copied from the parent P1, we get alleles B, G, F, and A from the parent P2. Starting from the second crossover site, which is the sixth gene, we copy alleles B and G as the sixth and seventh genes respectively. We then wrap around and copy alleles F and A as the first and second genes.

J. Cyclic Crossover (CX)

Cyclic crossover (CX): identifies a number of so-called cycles between two parent chromosomes.

1. **Step 1.** Child 1 gets the first gene of parent 1.
2. **Step 2.** In Parent 1, find gene equal to gene 1 in parent 2; check if child 1 contains this gene. If not, copy this gene to child 1 at the same place, let it be place *x*. Otherwise – go to step 5.
3. **Step 2.i.** In parent 1, find gene equal to gene x in parent 2; check if child 1 contains gene x. If not, copy gene x to child 1 at the same place, let it be place *x1*; repeat **Step 2.i** with x=x1. Otherwise – go to **Step 3**.
4. **Step 3.** Final step: once **Step 2** is completed, fill in empty genes of child 1 with corresponding values of parent 1.

Figure 3.10 illustrates CX iterations.

Parent 1	1	3	5	6	2	4	7
Parent 2	3	5	4	7	6	1	2
Offspring 1	**1**						
Offspring 2							

Parent 1	1	3	5	6	2	4	7
Parent 2	3	5	4	7	6	1	2
Offspring 1	**1**	**3**					
Offspring 2							

Parent 1	1	3	5	6	2	4	7
Parent 2	3	5	4	7	6	1	2
Offspring 1	**1**	**3**	**5**				
Offspring 2							

Parent 1	1	3	5	6	2	4	7
Parent 2	3	5	4	7	6	1	2
Offspring 1	**1**	**3**	**5**			**4**	
Offspring 2							

Parent 1	1	3	5	6	2	4	7
Parent 2	3	5	4	7	6	1	2
Offspring 1	**1**	**3**	**5**			**4**	
Offspring 2							

Fill up everything

Parent 1	1	3	5	6	2	4	7

Parent 2	3	5	4	7	6	1	2
Offspring 1	**1**	**3**	**5**	**7**	**6**	**4**	**2**
Offspring 2	**3**	**5**	**4**	**6**	**2**	**1**	**7**

Fig. 3.10 Cyclic crossover example, iterations from left to right

K. Evolving Crossover "Hot Spots"

A different approach, also inspired by nature, was taken by Schaffer and Morishima (1987). Their idea was to evolve not the order of bits in the string but rather the positions at which crossover was allowed to occur (crossover "hot spots"). Attached to each candidate solution in the population was a second string—a "crossover template"—that had a 1 at each locus at which crossover was to take place and a 0 at each locus at which crossover was not to take place. For example, 10011111:00010010 (with the chromosome preceding and the crossover template following the colon) meant that crossover should take place after the fourth and seventh loci in that string. Using an exclamation point to denote the crossover markers (each attached to the bit on its left), we can write this as 1001!111!1. Now, to perform multi−point crossover on two parents (say 1001!111!1 and 000000!00), the is mark the crossover points, and they get inherited along with the bits to which they are attached: Mutation acts on both the chromosomes and the attached crossover templates. Only the candidate solution is used to determine fitness, but the hope is that selection, crossover, and mutation will not only discover good solutions but also coevolve good crossover templates. Schaffer and Morishima found that this method outperformed a version of the simple GA on a small suite of function optimization problems. Although this method is interesting and is inspired by real genetics (in which there are crossover hot spots that have somehow coevolved with chromosomes), there has not been much further investigation into why it works and to what degree it will actually improve GA performance over a larger set of applications.

L. Arithmetic Crossover

Arithmetic crossover operator linearly combines two parent chromosomes vectors to produce two new offspring according to the equation:

Offspring 1 = a * Parent 1 + (1 - a) * Parent 2
Offspring 2 = (1 - a) * Parent 1 + a * Parent 2
where a is a random weighing factor chosen before each crossover operation.

Consider two parents (each of 4 float genes) selected for crossover:

Parent 1	0.3 1.4 0.2 7.4
Parent 2	0.5 4.5 0.1 5.6

Fig 3.11a Arithmetic Crossover (Before)

Now applying the two equations and assuming that the weighing factor a=0.7, we get two resulting offspring.
The possible set of offspring after arithmetic crossover would be:

Offspring 1	0.36 2.33 0.17 6.87
Offspring 2	0.402 2.981 0.149 5.842

Fig 3.11b Arithmetic Crossover (After)

M. Heuristic Crossover

Heuristic crossover operator uses the fitness value of two parent chromosomes to determine the direction of the search. The offspring are created according to the equations:

Offspring 1 = Best Parent + r * (Best Parent – Worst Parent)
Offspring 2 = Best Parent
where r is a random number between 0 and 1.
It is possible that offspring 1 will not be feasible. This can happen if r is chosen such that one or more of its genes fall outside the allowable upper or lower bounds. For this reason heuristic crossover has a user defined parameter n for the number of times to try and find an r that results in a feasible chromosome. If a feasible chromosome is not produced after n tries , the worst parent is returned as offspring 1.

N. Crossover with Reduced Surrogate
The reduced surrogate operator constrains 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.

O. Partially Matched Crossover (PMX)
Apart from always generating valid offspring, the PMX operator (Goldberg and Lingle, 1985) also preserves orderings within the chromosome. In PMX, two parents are randomly selected and two random crossover sites are generated. Alleles within the two crossover sites of a parent are exchanged with the alleles corresponding to those mapped by the other parent. It can be said that it is a crossover of permutations that guarantees that all positions are found exactly once in each offspring, ie 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 illustrate how PMX works.

- Consider the two strings shown below.
- Where the dots marl 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, the numbers that exchange places are 5 and 2, 6 and 3, and 7 and 10.
- The resulting offspring are as shown below:

Strings given

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

Partially matched crossover

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

Fig 3.12 Partially Matched Crossover (PMX)

Also, Fig gives a PMX example.

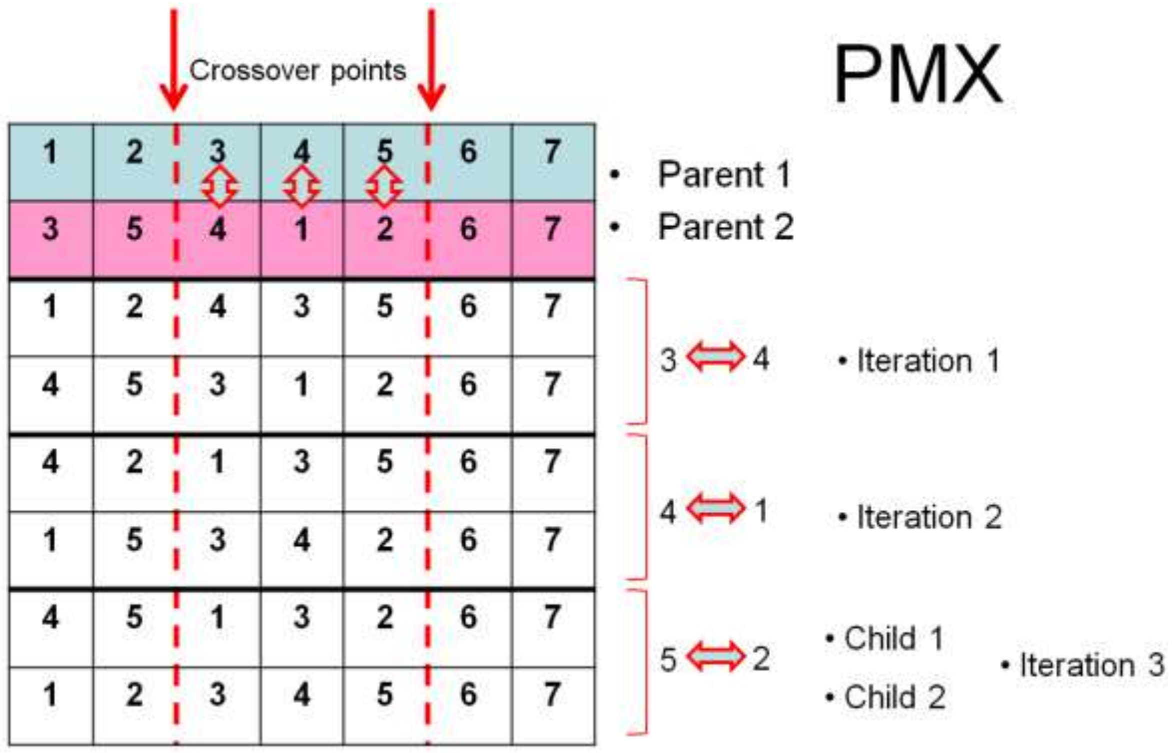

Fig 3.12. Partially-matched crossover example

P. Precedence Preservative Crossover (PPX)

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

- A vector of length Sigma (σ), sub i=1 to mi, representing the number of operations involved in the problem, is randomly filled with elements of the set {1,2}
- This vector defines the order in which the operations are successively drawn from parent 1 and parent 2.
- We can also consider the parent and offspring permutations as lists, for which the operations 'append' and 'delete' are defined.
- First we start by initializing an empty offspring.
- The leftmost operation in one of the two parents is selected in accordance with the order of parents given in the vector.
- After an operation is selected it is deleted in both parents.
- Finally the selected operation is appended to the offspring.
- This step 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 Fig. 3.13

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

Fig 3.13 Precedence Preservative Crossover (PPX)

Q. Cut and Splice

Another crossover variant, the "cut and splice" approach, results in a change in length of the children strings. The reason for this difference is that each parent string has a separate choice of crossover point.

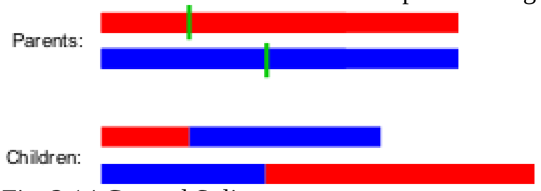

Fig. 3.14 Cut and Splice crossover

S. 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 parent's 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 survives to next generation.

VI. MUTATION

If we use a crossover operator, such as one-point crossover, we may get better and better chromosomes but the problem is, if the two parents (or worse, the entire population) has the same allele at a given gene then one-point crossover will not change that. In other words, that gene will have the same allele forever. Mutation is designed to overcome this problem in order to add diversity to the population and ensure that it is possible to explore the entire search space. In natural evolution, mutation is a random process where one allele of a gene is replaced by another to produce a new genetic structure. In GAs, mutation is randomly applied with low probability, typically in the range 0.001 and 0.01, and modifies elements in the chromosomes. Usually considered as a background operator, the role of mutation is often seen as providing a guarantee that the probability of searching any given string will never be zero and acting as a safety net to recover good genetic material that may be lost through the action of selection and crossover. Given that mutation is generally applied uniformly to an entire population of strings, it is possible that a given binary string may be mutated at more than one point. With non-binary representations, mutation is achieved by either perturbing the gene values or random selection of new values within the allowed range. In evolutionary strategies, mutation is the primary variation/search operator. Unlike evolutionary strategies, mutation is often the secondary operator in GAs, performed with a low probability. Mutation is a common operator used to help preserve diversity in the population by finding new points in the search pace to evaluate. When a chromosome is chosen for mutation, a random change is made to the values of some locations in the chromosome. Mutation adds new information in a random way to the genetic search process and ultimately helps to avoid getting trapped at local optima. It is an operator that introduces diversity in the population whenever the population tends to become homogeneous due to repeated use of reproduction and crossover operators. Mutation may cause the chromosomes of individuals to be different from those of their parent individuals. Mutation in a way is the process of randomly disturbing genetic information. They operate at the bit level; when the bits are being copied from the current string to the new string, there is probability that each bit may become mutated. This probability is usually a quite small value, called as mutation probability pm. A coin toss mechanism is employed; if random number between zero and one is less than the mutation probability, then the bit is inverted, so that zero becomes one and one becomes zero. This helps in introducing a bit of diversity to the population by scattering the occasional points. This random scattering would result in a better optima, or even modify a part of genetic code that will be beneficial in later operations. On the other hand, it might produce a weak individual that will never be selected for further operations.

The need for mutation is to create a point in the neighborhood of the current point, thereby achieving a local search around the current solution. The mutation is also used to maintain diversity in the population.

For example, the following population having four eight bit strings may be considered:

01101011
00111101
00010110
01111100.

It can be noticed that all four strings have a 0 in the left most bit position. If the true optimum solution requires 1 in that position, then crossover operator described above will be able to create 1 in that position. The inclusion of mutation introduces probability pm of turning 0 into 1. The crossover operator recombines good sub-strings from good strings together, hopefully, to create a better sub-string. The mutation operator alters a string locally expecting a better string. Even though none of these claims are guaranteed and/or tested while creating a string, it is expected that if bad strings are created they will be eliminated by the reproduction operator in the next generation and if good strings are created, they will be increasingly emphasized. Further insight into these operators, different ways of implementations and some mathematical foundations of genetic algorithms can be obtained from GA literature. Application of these operators on the current population creates a new population. This new population is used to generate subsequent populations and so on, yielding solutions that are closer to the optimum solution. The values of the objective function of the individuals of the new population are again determined by decoding the strings. These values express the fitness of the solutions of the new generations. This completes one cycle of genetic algorithm called a generation. In each generation if the solution is improved, it is stored as the best solution. This is repeated till convergence.

After recombination and mutation, the individual strings are then, if necessary, decoded, the objective function evaluated, a fitness value assigned to each individual and individuals selected for mating according to their fitness, and so the process continues through subsequent generations. In this way, the average performance of individuals in a population is expected to increase, as good individuals are preserved and bred with one another and the less fit individuals die out. The GA is terminated when some criteria are satisfied, e.g. a certain number of generations, a mean deviation in the population, or when a particular point in the search space is encountered. Mutation rate determines the probability that a mutation will occur. Mutation is employed to give new information to the population (uncover new chromosomes) and also prevents the population from becoming saturated with similar chromosomes, simply said to avoid premature convergence. Large mutation rates increase the probability that good schemata will be destroyed, but increase population diversity. The best mutation rate is 'application dependent'. For most applications, mutation rate is between 0.001 and 0.1, while for automated circuit design problems, it is usually between 0.3 and 0.8. Mutation is generally considered to be a background operator that ensures that the probability of searching a particular subspace of the problem space is never zero. This has the effect of tending to inhibit the possibility of converging to a local optimum, rather than the global optimum. The commonest ones out of the many mutation operators are:

- Single Point Mutation
- Multi Point Mutation
- Bit Flip Mutation
- Interchanging Mutation
- Reversing Mutation
- Mutation Probability
- Reorder Mutation
- Uniform Mutation

A. Single Point Mutation

A commonly used method for mutation is called single point mutation. Single gene (chromosome or even individual) is randomly selected to be mutated and its value is changed depending on the encoding type used.

Consider two parents selected for mutation

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

Fig 3.15a Single Point Mutation (Before)

The mutated offspring produced after changing or inverting the value of the chosen gene as 0 to 1 and 1 to 0 are

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

Fig 3.15b Single Point Mutation (After)

B. Multi Point Mutation

Multi genes (chromosomes or even individuals) are randomly selected to be mutated and there values are changed depending on the encoding type used, as shown in Fig. 3.14a and Fig. 3.14b.

Here are two parents selected for mutation

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

Fig 3.16a Multi Point Mutation

The mutated offspring produced are

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

Fig 3.16b Multi Point Mutation

C. Bit-Flip Mutation (Flipping)

In bit flip mutation, each chosen bit in a parent binary string (chromosome) is changed (a 0 is converted to 1, and a 1 is converted to 0) to produce offspring. A parent 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) offspring chromosome is produced. This is illustrated in the Fig. 3.15

Parent	**1** 1 1 **0** 1 1 0 **0** 0 1
Mutation chromosome	**0** 1 0 **1** 1 0 0 **1** 0 0
Offspring	**0** 1 1 **1** 1 1 0 **1** 0 1

Fig 3.17 Bit-Flip Mutation

D. Interchanging Mutation

Two random positions of the string are chosen and the bits corresponding to those positions are interchanged. This is shown in Fig. 3.15

Parent	1 **0** 1 1 0 **1** 0 1
Offspring	1 **1** 1 1 0 **0** 0 1

Fig 3.18 Interchanging Mutation

E. Reversing Mutation

A random position is chosen and the bits next to that position are reversed and child chromosome is produced. This is shown in Fig. 3.16.

Parent	0 1 1 0 1 1 **0** 1 0 1
Offspring	0 1 1 0 1 1 0 **0 1 0**

Fig 3.19 Reversing Mutation

F. Mutation Probability
The important parameter in the mutation technique is the mutation probability (P_m). The mutation probability 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 permutation is performed, one or more parts of a chromosome are changed. If mutation probability is 100%, whole chromosome is changed, if is 0%, nothing is changed. Mutation generally prevents the GA from falling into local extreme.

G. Reorder Mutation
This swaps the positions of pair of bits or genes which are selected randomly to increase diversity in the decision variable space.

H. Uniform Mutation
The mutation operator replaces the value of the chosen gene with a uniform random value selected between the user-specific upper and lower bounds for that gene.

VII. REPLACEMENT

Replacement, which is the last stage of any breeding cycle tends to be the most important stage. Replacement determines the current members of the population, ie old parents or offspring been produced should be replaced by new offspring if any. Two parents are drawn from a fixed size population, they breed two children, but not all four can return to the population, so two must be replaced. The technique used to decide which individual stay in a population and which are replaced in on a par with the selection in influencing convergence. Basically, there are two kinds of methods for maintaining the population: generational updates and steady state updates. The basic generational update scheme consists in producing N children from a population of size N to form the population at the next time step (generation), and this new population of children completely replaces the parent selection. Clearly this kind of update implies that an individual can only reproduce with individuals from the same generation. Derived from forms of generational update are also used like ($\lambda+\mu$)-update and (λ,μ)-update. This time from a parent population of size μ, a little of children is produced of size $\lambda\geq\mu$. Then the μ best individuals from either the offspring population or the combined offspring and parent populations (for (λ,μ)- and ($\lambda+\mu$)-update respectively), from the next generation.

In a steady state update, new individuals are inserted in the population as soon as they are created, as opposed to the generational update where an entire new generation is produced at each time step. The insertion of a new individual usually necessitates the replacement of another population member. The individual to be deleted can be chosen as the worst member of population, (it leads to a very strong selection pressure), or as the oldest member of the population, but those method are quite radical: Generally steady state updates use an ordinal based method for both the selection and the replacement, usually a tournament method. Tournament replacement is exactly analogous to tournament selection except the less good solutions are picked more often than the good ones. A subtile alternative is to replace the most similar member in the existing population. When selecting which members of the old population should be replaced the most apparent strategy is to replace the least fit members deterministically. Thus, for an individual to survive successive generations, it must be sufficiently fit to ensure propagation into future generations. Some of the most common replacement techniques are outlined below.

A. Random Replacement
The children replace two random chosen individuals in the population. The parents are also candidates for selection. This can be useful for continuing the search in small populations, since weak individuals can be introduced into the population.

B. Weak Parent Replacement
In weak parent replacement, a weaker parent is replaced by a strong child. With the four individuals, only the fittest two, parent or child return to the population. This process improves the overall fitness of the population when paired with a selection technique that selects both fit and weak parent for crossing, but if weak individuals and discriminated against in selection, the opportunity will never raise to replace them.
C. Both Parents Replacement

Both parents replacement is when the child replaces the parent. In this case, each individual only gets to b reed once. As a result, the population and genetic material moves around but leads to a problem when combined with a selection technique that strongly favours fit parents: the fit breed and then are disposed off.

VIII. SEARCH TERMINATION (CONVERGENCE CRITERIA)

Because the GA is a stochastic search method, it is difficult to formally specify convergence criteria. As the fitness of a population may remain static for a number of generations before a superior individual is found, the application of conventional termination criteria becomes problematic. A common practice is to terminate the GA after a pre-specified number of generations and then test the quality of the best members of the population against the problem definition. If no acceptable solutions are found, the GA may be restarted or a fresh search initiated. The various stopping conditions are listed as follows:

- ***Maximum generations-*** the genetic algorithm stops when the specific number of generations have evolved.
- ***Elapsed time-*** The genetic process will end when a specific time has elapsed.
  Note: If the maximum number of generation has been reached before the specific time has elapsed, the process will end.
- ***No change in fitness-*** The genetic process will end if there is no change to the population's best fitness for a specific number of generations.
  Note: If the maximum number of generation has been reached before the specific number of generations with no changes has been reached, the process will end.
- ***Stall generations-*** The algorithm stops if there is no improvement in the subjective function for a sequence of consecutive generations.
- ***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.

A. 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.

B. 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.

C. 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.

D. 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.

IX. HOW GENETIC ALGPRITHM WORK

Although genetic algorithm are simple to describe and program, their behavior can be complicated, and many open questions exist about how they work and for what types of problems they are best suited. Much work has been done on the theoretical foundations of GAs. The traditional theory of GAs assumes that, at a very general level of description, GAs work by discovering, emphasizing, and recombining good building blocks of solutions in a highly parallel fashion. The idea here is that good solutions tend to be made up of good building blocks, combinations of bit values that confer higher fitness on the string in which they are present. Holland (1975) introduced the notion of schemas (or schemata) to formalize the informal notion of building blocks. A schema is a set of bit strings that can be described by a template made up of ones, zeros and asterisks; the asterisks representing wild cards (don't cares).

For example, the schema $H = 1 * * * * 1$ represents the set of all 6-bit strings that begin and end with 1 where h stands for 'hyperplane'. *H* is used to denote schemas because schemas define hyperplanes; planes of various dimensions in the *l* dimensional space of length-*l* bit strings. The strings that fit this template (e.g., 100111 and 110011) are said to be *instances* of *H*. The schema *H* is said to have two defined bits (non-asterisks) or, equivalently, to be of *order* 2. Its *defining length* (the distance between its outermost defined bits) is 5.
Here the term schema is used to denote both a subset of strings represented by such a template and the template itself.

X. WHEN TO USE A GENETIC ALGORITHM

The GA literature describes a large number of successful applications, but there are also many cases in which GA performs poorly. Given a particular potential application, how do you know if GA is a good method to use? There is no rigorous answer; though many researchers share the intuition that if the space to be searched is known not to be perfectly smooth and uni-modal (consist of a single smooth hill), or is not well understood, or if the fitness function is noisy, and if the task does not require a global optimum to be found, that is, if quickly finding a sufficiently good solution is enough, a GA will have a good chance of being competitive with or surpassing other weak methods. If a space is not large, then it can be search exhaustively, whereas a GA might converge on a local optimum rather than on the global best solution. If the space is smooth or uni-modal, a gradient-ascent algorithm such as steepest-ascent hill climbing will be much more efficient than the GA in exploiting the space's smoothness. If the space is well understood, search methods using domain specific heuristics can often be design to outperform any general-purpose method such as a GA. If the fitness function is noisy (e.g., if it involves error-prone measurements from a real-world process such as vision system of a robot), a one-candidate-solution-at-a-time search method such as simple hill climbing might be irrecoverably led astray by the noise, but GAs, since they work by accumulating fitness statistics over many generations, are thought to perform robustly in the presence of a small amounts of noise. These institutions, of course, do not rigorously predict when a GA will be an effective search procedure competitive with other procedures. A GA's performance will depend very much on details such as the method for encoding candidate solution, the operators, the parameter settings, and the particular criterion for success.

XI. BUILDING BLOCK HYPOTHESIS

A genetic algorithm seeks near-optimal performance through the juxtaposition of short, low-order, high-performance schemata, called the building blocks (BBs). The building block hypothesis one of the most important criteria of how a genetic algorithm works. The importance of building blocks (BBs) and the role in the working of GAs have long been recognized (Holland, 1975; Goldberg, 1989). Furthermore, the following six for s GA success have been proposed (Goldberg, Deb and Clark, 1992).

- Identify GAs which are the processing-building blocks.
- Ensure an adequate initial supply of raw BBs.
- Ensure growth of superior BBs.
- Ensure the mixing of BBs.
- Ensure good decisions among competing BBs and
- Solve problems with bounded BB difficulty.

One of the most important conditions is to make sure that the GA is well supplied with a sufficient supply of the BBs required to solve a given problem. It is also equally important that proportion of the good ones in the population grow. The first and the second task, that is guaranteeing the increase in market share of good BBs in a population has been recognized by Goldberg, Sartry and Laloza, (2001). The usual approach in achieving this is the schema theorem (Holland and De Jong (1997)).

XII. THE SCHEMA THEOREM

Considering proportionate selection, single-point crossover, and no mutation the schema theorem may be written as follows:

$$E\left(m\left(H,t+1\right)\right) > m\left(H,t\right)\frac{f\left(H,t\right)}{f\left(t\right)}\left\{1-P_c\frac{\delta\left(H\right)}{L-1}\right\}$$

where:

$m\left(H,t+1\right)$ is the expected number of individuals that represents the schema *H* at generation *t + 1*.

$m\left(H,t\right)$ is the expected number of individuals that represents the schema *H* at count generation *t*.

$f\left(H,t\right)$ is the average fitness value of the individual containing schema *H* at generation *t*.

$\bar{f}\left(t\right)$ is the average fitness of the population at generation *t*.

P_c is the crossover probability.

$\delta\left(H\right)$ is the defining length defined as the distance between the outer-most fixed positions of the schema.

L is the string length.

Inspection of the schema theorem and an analysis of proportionate selection and single-point crossover indicates that the term $m\left(H,t\right)\frac{f\left(H,t\right)}{f\left(t\right)}$ account for the selection and the term $\left\{1-P_c\frac{\delta\left(H\right)}{L-1}\right\}$ account for crossover operation.

It should be noted that the term representing the selection operation is exact and inequality occurs due to crossover operation. Some factors like crossover between identical individuals (self-crossover) are neglected.
The schema theorem tells us that the proportion of schemata increases when they have above average fitness and relatively low crossover disruption. However, the schema theorem as given by the equation above is restricted to proportionate selection and one-point crossover. This concern can be eliminated by identifying the characteristic form of schema theorem and substituting appropriate terms for other selection schemes and genetic operators. However, the generalized schema theorem can alternatively be written in the form:

$$E\left(m\left(H,t+1\right)\right) \geq \frac{f\left(H,t\right)}{f\left(t\right)}m\left(H,t\right)\left\{1-P_c\frac{\delta\left(H\right)}{L-1}\right\}\left(1-P_m\right)^{0(H)}$$

where;

$\left(1-P_m\right)^{0(H)}$ account for mutation operations and

$0\left(H\right)$ is the number of fixed bits in the schema.

These particular schemata are called building blocks and its applications are as follows:

- It provides some tools to check whether a given representation is well-suited to a genetic algorithm.
- The analysis of nature of the good schemata gives a few ideas on the efficiency of genetic algorithm.

XIII. NO-FREE-LUNCH THEOREM

The No-Free-Lunch theorem states that without any structural assumption on a search or optimization problem, no algorithm can perform better than blind search. To achieve a performance evaluation for an algorithm, it is not sufficient to demonstrate its better performance on a given set of functions. Instead of this, the diversity of an algorithm should be considered. The total number of possible algorithms as well should be computed with the number of algorithms instances that a random search or a population based algorithm can have. The question now is, how many different algorithms can be provided by such an algorithm class, and how does this number behave with respect to the total number of possible algorithm.

The answer gives us a ranking for algorithms according to their smaller or larger number of instances. It comes out that by such a ranking, random search is worst, while evolutionary approaches are (at least theoretically) able to provide any search sequence that is possible which implies that, population-based algorithms are principally able to cover the set of all possible algorithms. The No-Free-Lunch theorem also provides information on the following:

- The geometric interpretation of what it means for an algorithm to be well matched to a problem.
- Brings insight provided by information theory into the search procedure.
- It provides that independent of the fitness function one cannot (without prior domain knowledge) successfully choose between two algorithms based on their previous behavior.

XIV. THE FLOYD-WARSHALL ALGORITHM

The Floyd-Warshall algorithm compute the all pairs shortest path matrix. It uses a vectorised version of the Floyd-Warshall function. As in the dynamic programming algorithm, we assume that the path is represented by an *n x n* matrix with the weights of the edges.

$$w_{ij} = \begin{cases} 0 & if\ i = j \\ w_{ij} & if\ i \neq j\ and\ (i, j) \in E \\ \infty & if\ i \neq j\ and\ (i, j) \notin E \end{cases}$$

Output format : An *n x n* distance $D = [d_{ij}]$ where d_{ij} is the distance from vertex *i* to *j*.
The objective is to find an estimate for the infinitesimal values using the Algorithm.

Fig 3. . The Floyd-Warshall Algorithm
The algorithm for the Floyd-Warshall is as follows:

```
Floyd-Warshall (w,n)
{for $i = 1$ to $n$ do                initiate
    for $j = 1$ to $n$ do
        { $D^o[i, j] = w[i, j]$;
         $pred[i, j] = nil$;}
for $k = 1$ to $n$ do                 Dynamic programming
    for $i = 1$ to $n$ do
        for $j = 1$ to $n$ do
            if $(d^{k-1}[i,k] + d^{k-1}[k, j] < d^{k-1}[i, j])$
                $d^k[i, j] = d^{k-1}[i,k] + d^{k-1}[k, j]$;
                $pred[i, j] = k$;
            else $d^k[i, j] = d^{k-1}[i, j]$;
return $d^n[1..n,1..n]$ }
```

XV. APPLICATIONS OF GENETIC ALGORITHMS

Genetic algorithms (GAs) are adaptive methods which may be used to solve search and optimization problems. The power of GAs comes from the fact that the technique is robust and can deal successfully with a wide range of problem areas, including those which are difficult for other methods to solve. Therefore, the main ground for GAs is in difficult areas where no such solving techniques exist. Even where existing techniques work well, improvements can be made by mixing them with GAs.
GAs in various forms are implemented to wide range of problems including the following:

- ***Optimization:*** GAs have been used in a wide variety of optimization tasks, including numerical optimization and combinatorial optimization problems such as circuit design and job shop scheduling.

- ***Automatic Programming:*** GAs have been used to evolve computer programs for special tasks and to design other computational structures cellular automata and sorting networks.

- ***Machine and robot learning****:* GAs have been used for many machine learning applications, including classification and prediction tasks such as the prediction of dynamical systems, weather prediction , and prediction of protein structure. GAs have also been used to design neural networks, and to evolve rules for learning classifier systems or symbolic production systems and to design and control robots

- ***Economic models:*** GAs have been used to model processes of innovation, the development of bidding strategies, and the emergence of economic markets.

- ***Immune system models:*** GAs have been used to model various aspects of the natural immune system including somatic mutation during an individual’s lifetime and the discovery of multi-gene families during evolutionary time

- ***Ecological models:*** GAs have been used to model ecological phenomena such as biological arms races, host-parasite co-evolution, symbiosis, and resource flow in ecologies.

- ***Population genetics models:*** GAs have been used to study questions in population genetics, such as “under what conditions will a gene for recombination be evolutionarily viable (or practicable) ?”

- ***Interactions between evolution and learning:*** GAs have been used to study how individual learning and species evolution affect one another.

- ***Models of social systems:*** GAs have been used to study evolutionary aspects of social systems, such as the evolution of cooperation, the evolution of communication, and trail-following behavior in ants.

This list is by no means exhaustive, but it gives a flavor of the kinds of things for which GAs have been used, both for problem-solving and for modeling.

XVI. ADVANTAGES AND DISADVANTAGE OF GENETIC ALGORITHMS

Perhaps it is not obvious why such an algorithm should lead to accurate solutions for optimization problems. Crossover is a crucial aspect of any genetic algorithm, but it may seem that it will dramatically change parents with a high fitness function so that they will no longer be fit. However, this is not the case. As in biology, crossover can lead to new combinations of genes which are more fit than any in the previous generations. Other offspring will be created which are less fit but these will have a low probability of being selected to go on to the next generation. Creating new variants is the key to genetic algorithms, as there is a good chance of finding better solutions. This is why mutation is also a necessary part of the genetic algorithms. It will create offspring which would not have arisen otherwise, and may lead to a better solution. Other optimization algorithms have the disadvantage that some kind of initial guess is required and this may bias the final result. GAs on the other hand only require a search range, which need only be constrained by prior knowledge of the physical properties of the system. Effectively they search the whole of the solution space, without calculating the fitness function at every point. This can help avoid a danger in any optimization problem which is being trapped in local maxima or minima. There are two main reasons for this:

- The initial population, being randomly generated, will sample the whole of the solution space, and not just a small area.
- Variation inducing tactics, i.e. crossover and mutation, prevent the algorithm being trapped in one part of the solution space.

Genetic algorithms can be employed for a wide variety of optimization problems. They perform very well for large scale optimization problems which may be very difficult or impossible to solve by other traditional methods.

Implementation of the genetic algorithm usually does not require much knowledge about the structural properties of the problem and the algorithm can be easily coded. Often genetic algorithms produce fairly good solutions. The disadvantage of genetic algorithms is that it, sometimes; have trouble finding the exact global optimum because there is no guaranty to find best solution. Genetic algorithm may be less efficient (in terms of the running time and the accuracy of the solution) than problem-specific approaches. Another drawback is that GAs require large number of response (fitness) function evaluations depending on the number of individuals and the number of generations. Therefore, genetic algorithms may take long time to evaluate the individuals.

XVII. THE TRADITIONAL SEARCH METHODOLOGIES:

The optimization methods are formally divided to three methods as follows:

- Mathematical optimization methods (calculus-based methods)
- Enumerative optimization methods
- Stochastic optimization methods

Many studies were down on calculus based methods during the last decades. These methods are divided to the directed and undirected methods. In direct methods, the local extremis is searched based on a set of nonlinear equations. Really these equations are derived from the first derivation of function when it is equaled to zero. Versus in the directed methods, the local extremis values are determined by moving and jumping on the functions and along the local gradient. However both of them have some serious problems such as local answers, time consumption efficiency and applicability for all real word problems. Enumerative methods are utilized in different sizes and forms in optimization problems. The initial idea of this methodology is very simple. In this method the studied area was divided to specific parts and then monitoring the change of objective function in each node (divided point) of studied area. In the other word, a network is designed for studied area and objective variability is monitored along the time. This methodology has some problems due to low efficiency. The stochastic search methodology is more desirable because it has lower problem relative to the previous methods. It can be expected that the stochastic methodologies act more appropriate than numerical methods. The genetic algorithm methods can be placed in this framework. The traditional search methodologies are not robust. But this does not imply that they are not appropriated. Rather they can be operated in some application very well. However they have many deficiency when are used in complex problems.

XVIII. DIFFERENCES BETWEEN GENETIC ALGORITHM AND OTHER OPTIMIZATION METHODS (TRADITIONAL METHODS):

In order to show which properties of genetic algorithm are better than traditional cousins, some properties of genetic algorithm is noted. In many traditional methods, the movement from one point to the other point in decision space is made with caution and using the transition rules. Such movement has some shortcoming, because it causes the final answer to be placed on a no real extremum point when the search space has many extremums. But Genetic algorithm created a wide search space. Many strings concurrently and in parallel search the extremum points of decision space. This makes the probability of no correct point finding decreases relative to traditional methods.

Many of traditional search methods need more auxiliary information than genetic algorithms in order to work better. For example the gradient methods need derivatives in order to be able to climb the current peak. However the genetic algorithm does not require any auxiliary knowledge. They only need a payoff function (objective function) for a proper and effective search. These properties cause the genetic algorithms find more legitimatization relative to the other methods.

Unlike the other methods, GAs utilizes probabilistic transition rules to direct their search. However, the use of probability dose not suggest that the method is some simple random search. Really the genetic algorithm utilizes the random search to guide a search toward the regions space with likely improvement.

Finally, the sum of properties as coding, search into a population, independency from the auxiliary knowledge and stochastic operators have exchanged the Genetic algorithm to a robust technique.

XIX. CONCLUSION

- The genetic algorithm works with a coding of the parameters and not the actual parameters of the problem i.e., GA works with the coding of solutions and not with the solution itself (except in where real-valued individuals are used)

- Whereas other optimization techniques search from a single point, the genetic algorithm searches a set of points which is the whole population and not one point. This is to say that it uses population of solutions rather than a single solution from searching resulting to the robustness of genetic algorithm, hence improving the chance of reaching the global optimum and also helps in avoiding local stationary point.

- Genetic algorithms do not require derivative information or other auxiliary knowledge; only the objective function and corresponding fitness levels influence the directions of search. This helps to be able to be applied to any kind of continuous or discrete optimization problem knowing that the key point to be performed here is to identify and specify a meaningful decoding function.

- The genetic uses some probabilistic transition rules, not deterministic rules.

References

A. Leon and J. Pozo, Using a genetic algorithm to study properties of minimum energy states and geometrical frustration in artificial ‘‘spin ice’’ systems, Journal of Magnetism and Magnetic Materials 320, pp. 210–216, (2008).

A. A. Ghanbari, A. Broumandnia, H. Navidi, A. Ahmadi, Brain Computer Interface with Genetic Algorithm, International Journal of Information and Communication Technology Research Vol. 2 No. 1, (2012).

A. Ghosh and S. Dehuri, Evolutionary algorithms for Multi-Criterion Optimization: A Survey, International Journal of Computing and Information Sciences, vol.2, pp 35-62, (2004)

Arifovic, J., Evolutionary algorithms in macroeconomic models. Macroeconomic Dynamics 4 (5): pp 373-414, (2000).

Birchenhall, C., N. Kastrinos, and J. Metcalfe, Genetic algorithms in evolutionary modeling, Journal of Evolutionary Economics 7 (4): pp 375-393, (1997).

Bremermann, H. J., The evolution of intelligence. The nervous system as a model of its environment, Technical Report No. 1, Department of Mathematics, University of Washington, Seattle, WA, (1958).

Carlsson, C., and E. Turban, DSS: Directions for the next decade. Decision Support Systems 40 (1): pp1-8, (2002).

Cassaigne, N., Aversi, R., G. Dosi, G. Fagiolo, M. Meacci, and C. Olivetti, Demand dynamics with socially evolving preferences. IIIASA Working Paper, Luxemburg, Austria, (1997).

C. Darwin, The Origin of Species, Dent Gordon, London, (1973).

C. Basnet, A. Weintraub, A Genetic Algorithm for a Bicriteria Supplier Selection Problem,(2001)

Chia-Feng Juang. A TSK-type recurrent fuzzy network for dynamic systems processing by neural network and genetic algorithm, IEEE Transactions on Fuzzy Systems, 10(2), pp. 155-170, (2002)

Chuan-Yin T. and Li-Xin G. Research on Suspension System Based on Genetic Algorithm and Neural Network Control, The Open Mechanical Engineering Journal, pp. 72-79, (2009).

Dosi, G., L. Marengo, A. Bassanini, and M. Valente., Norms as emergent properties of adaptive learning. Journal of Evolutionary Economics 9 (1): pp 5-26, (1999).

D. B. Fogel, ‘Evolutionary Computation: Toward a New Philosophy of Machine Intelligence’, IEEE Press, Piscataway, NJ, (1995).

D. B. Fogel, ‘Evolving Artificial Intelligence’, Ph.D. Thesis, University of California, San Diego, CA, (1992).

D. E. Goldberg, Sizing populations for serial and parallel genetic algorithms. In Schaffer, J. D. (editor) Proceedings of the Third International Conference on Genetic Algorithms. San Mateo, CA: Morgan Kaufmann Publishers Inc. (1989).

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

D. Goldberg and R. Lingle, Alleles, loci and traveling salesman problem, In J.J. Grefenstette, editor, Proceedings of International Conference on GAs, Lawrence Erlbaum, (1985).

D. E. Goldberg, B. Korb, K. Deb, Messy Genetic Algorithms: Motivation, Analysis, and First Results, Complex Systems 3, pp. 493-530, (1989).

D. E. Goldberg, Genetic algorithms in search, optimization, and machine learning, Addison Wesley Longman, Inc., ISBN 0-201-15767-5, (1989).

Dawid, H. Adaptive learning by genetic algorithms: Analytical results and application to economic models. Berlin: Springer, (1999).

Davis, L., Applying algorithms to epistatic domains, in: *Proc. Int. Joint Conf. on Artifical Intelligence*, pp. 162–164, (1985).

Davis, Lawrence (editor), *Genetic Algorithms and Simulated Annealing*. London: Pittman, (1987).

Davis, Lawrence, *Handbook of Genetic Algorithms*. New York: Van Nostrand Reinhold, (1991).

De Jong K., An Analysis of the Behaviour of a Class of Genetic Adaptive Systems, PhD Dissertation, Department of Computer and Communication Sciences, University of Michigan, Ann Arbor, (1975).

E. Alba and J. M. Troya, A Survey of Parallel Distributed Genetic Algorithms, Journal of Information and Operations Management
ISSN: 0976–7754 & E-ISSN: 0976–7762 , Vol. 3, pp 38-42, (2012).

E. Correa, R. Goodacre, A genetic algorithm-Bayesian network approachfor the analysis of metabolomics and spectroscopic data: application to the rapid identification of Bacillus spores and classification of Bacillus species, Correa and Goodacre BMC Bioinformatics, (2011)

Forrest, S., Genetic algorithms: Principles of natural selection applied to computation, *Science* **261**:872–878,(1993).

F. Oliveira, M.R. Almeida and S. Hamacher, Genetic algorithms applied to scheduling and optimization of refinery operations, **(2001).**

Frenke, R., Co-evolution and stable adjustment in the cobweb model, Journal of Evolutionary Economics 8 (4): pp 383-406, (1998).

Fraser, A. S., Simulation of genetic systems by automatic digital computers II: Effects of linkage on rates under selection, Austral. J. Biol. Sci. 10: pp 492–499, (1957).

Forel, D. B., and Atmar, W., eds., Proceedings of the second on Evolutionary Programming. Evolutionary Programming Society, (1993)

Fogel L. J., Autonomous automata. Industrial Research 4: pp 14–19, (1962).

Fogel L. J., Owens A. J. and Walsh M. J., Artificial Intelligence through Simulated Evolution, Wiley, New York, (1966).

G. Kannan, P. Sasikumar and K. Devika, A genetic algorithm approach for solving a closed loop supply chain model: A case battery recycling Journal of Applied Mathematics, vol.34, pp. 655-670, (2010)

Goldberg, D. E., Deb, K., and Korb, B. Messy genetic algorithms revisited: Studies in mixed size and scale. Complex Systems 4. pp. 410-470, (1990)

Goldberg, D. E. Genetic Algorithms and the minimal deceptive problem. In L. D. Davis, ed., Genetic Algorithms and Simulated Annealing. Morgan Kaufmann, (1987)

Goldberg, D. E. A note on Boltzman tournament selection for genetic algorithms and population-oriented simulated annealing. Complex Systems 4, pp. 445-460.

Goldberg, D. E. Genetic Algorithms and Walsh Functions: Part I, A gentle introduction. Complex Systems 3: pp. 130-155 (1998).

Goldberg, D. E., Genetic algorithms and Walsh functions: Part II, deception and its analysis, *Complex Syst.* **3**:153–171, (1989a).

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

Goldberg, D. E., Sastry, K. and Latoza, T., On the supply of building blocks, in: *Proc. of the Genetic and Evolutionary Computation Conf.*, pp. 336–342, (2001).

Goldberg, D. E. and Lingle, R., Alleles, loci, and the TSP, in: *Proc. 1st Int. Conf. on Genetic Algorithms*, pp. 154–159, (1985).

Gen, M. and Cheng, R., Genetic algorithms and engineering optimization, John Wiley, New York, (2000).

Holland, John H., Escaping brittleness: The possibilities of general-purpose learning algorithms applied to parallel rule-based systems. In Michalski, Ryszard S., Carbonell, Jaime G. and Mitchell, Tom M. Machine Learning: An Artificial Intelligence Approach, Los Altos, CA: Morgan Kaufman, vol II. P. 593-623, (1986).

Hong Y. Y. and Li C. . Genetic algorithms based Economic Dispatch for Cogeneration Units Considering Multiplant Multibuyer Wheeling, IEEE Power Engineering Review, 22(1), pp. 66-69, (2002).

H.-P. Schwefel, 'Numerical Optimization of Computer Models', Wiley, Chichester, (1981).

H. G. Beyer and H. P. Schwefel, Evolution strategies: A comprehensive introduction, Natural Computing 1, pp. 3-52, (2002).

H. P. Schwefel. Evolutions strategies and Numeric Optimization, Dr.-Ing. Thesis, Technical University of Berlin, Department of Process Engineering, (1975).

H. R. Kanan, M. H. Moradi, A Genetic Algorithm based Method for Face Localization and pose Estimation, 3rd International Conference: Sciences of
Electronic, Technologies of Information and Telecommunications, (2005)
Hidalgo, J., F. Soltero, D. Bodas-Sagi, and P. Fernandez-Blanco, Technical market indicators optimization using evolutionary algorithms. Paper presented at the thirteenth IEEE International conference on evolutionary computation, Atlanta, Georgia, USA, (2008).

Holland J. H., Outline for a logical theory of adaptive systems. JACM 9: pp 297–314, (1962).

J. Blazewicz, M. Szachniuk and A. Wojtowicz, Evolutionary algorithm for a reconstruction of NOE paths in NMR spectra of RNA chains, Bulletin Of The Polish Academy Of Sciences Technical Sciences, Vol. 52, No. 3, pp. 8-33, (2004).

J. H. Holland, 'Adaptation in Natural and Artificial Systems', MIT Press, Cambridge, MA, (1992).

J. H. Holland, Adaptation in Natural and Artificial Systems, The University of Michigan Press, Ann Arbor, Michigan, (1975).

Jina, N., E. Tsang, and J. Li., A Constraint-guided method with evolutionary algorithms for economic problems. Applied Soft Computing 9 (3): pp 924-935, (2009).

J. D. Schaffer and A. Morishima, "An adaptive crossover distribution mechanism for genetic algorithms," in *Proceeding of the Second International Conference on Genetic Algorithms*, pp. 36-40, (1987).

J.F. Gonçalves, J.M. Mendes, and M.C.G. Resende. A hybrid genetic algorithm for the job shop scheduling problem. *European Journal of Operational Research*, vol. 167, pp. 77-95, (2005).

Kinnear, Kenneth E. Jr. (editor), Advances in Genetic Programming. Cambridge, MA: MIT Press, (1994).

Koza, J. R., Genetic Programming: On the Programming of Computers by Means of Natural Selection. Cambridge, MA: The MIT Press, (1992).

Koza, J., Genetic Programming II: Automatic Discovery of Reusable Programs, The MIT Press,USA, (1994).

Krohling R. A. and Rey J. P. Design of optimal disturbance rejection PID controllers using genetic algorithm, IEEE Transactions on Evolutionary Computation, 5(1), pp. 78-82, (2001).

L. J. Fogel, A. J. Owens, and M. J. Walsh, 'Artificial Intelligence through Simulated Evolution', Wiley, New York, (1966).

Muhlenbein, H. and Schlierkamp-Voosen, D., Predictive models for the breeder genetic algorithm: I. continous parameter optimization, *Evol. Comput.* **1**, pp. 25–49, (1993).

Mora, M., G. Forgionne, J. Gupta, L. Garrido, F. Cervantes, and O. Gelman, “A strategic review of the intelligent decision-making support systems research: The 1980-2004 period”. In Intelligent decision-making support systems: Foundations, applications and challenges, eds., pp. 441-462. Germany: Springer-Verlag, (2006)

Marczyk, A., Genetic algorithms and evolutionary computation, (2004).

M. Younes, M. Rahli and L. A. Koridak, Economic Power Dispatch Using Evolutionary Algorithm, Journal of Electrical Engineering, Vol. 57, No. 4, pp 211–217, (2006).

M. Sedighizadeh, and A. Rezazadeh, Using Genetic Algorithm for Distributed Generation Allocation to Reduce Losses and Improve Voltage Profile, World Academy of Science, Engineering and Technology 37, (2008).

Manish C. T, Yan F., and Urmila M. D., Optimal Design of Heat Exchangers: A Genetic Algorithm Framework, Ind. Eng. Chem. Res., 1999, 38 (2), 456-467 (1998).

M. H. Khorshid and H. A. Hassan, An Economic Optimization-Oriented Decision Support Model Based On Applying Single-Objective Evolutionary Algorithms To Computable General Equilibrium Models, Working Paper No. 9, pp. 15-39, (2010).

M. H. Khorshid and H. A. Hassan, An Economic Optimization-Oriented Decision Support Model Based On Applying Single-Objective Evolutionary Algorithms To Computable General Equilibrium Models, Working Paper No. 9, pp 4-12, (2010).

M. Hosseinzadeh and E. Roghanian, An Optimization Model for Reverse Logistics Network under Stochastic Environment Using Genetic Algorithm, International Journal of Business and Social Science Vol. 3 No. 12, (2012)

M. Kumar, M. Husian, N. Upreti and D. Gupta, Genetic Algorithm: Reviewand Application, International Journal of Information Technology and Knowledge Management, Vol. 2, No. 2, pp. 451-454, (2010)

N. Tyagi and R. G. Varshney, A Model To Study Genetic Algorithm For The Flowshop Scheduling Problem, Journal of Information and Operations Management, Vol 3, pp-38-42 (2012)

Osyczka, A. Evolutionary algorithms for single and multi-criteria design optimization. Germany: Physica Verlag, (2002).

Oliver, J. M., Smith, D. J. and Holland, J. R. C., A study of permutation crossover operators on the travelling salesman problem, in: *Proc. 2nd Int. Conf. on Genetic Algorithms*, pp. 224–230, (1987).

O’Reilly, U.M. and Oppacher, F., 'Program search with a hierarchical variable length representation: Genetic programming, simulated annealing and hill climbing', Davidor, Y. ,Schwefel, H.P. and Manner, R. (Ed.), *Parallel Problem Solving in Nature - PPSN III*, Vol.866 of Lecture Notes in Computer Science, Springer-Verlag, Germany, pp.397-406, (1994).

P.J.B. Hancock, An empirical comparison of selection methods in evolutionary algorithms. In T.C. Fogarty (ed.), *Evolutionary Computing: AISB Workshop, Leeds, UK, April 1994; Selected Papers.* Springer-Verlag, Berlin, pp. 80–94, (1994).

Po-Hung Chen and Hong-Chan Chang. Genetic aided scheduling of hydraulically coupled plants in hydro-thermal coordination, IEEE Transactions on Power Systems, 11(2), pp. 975-981, (1996)

R. Kumar, Blending Roulette Wheel Selection & Rank Selection in Genetic Algorithms, International Journal of Machine Learning and Computing, Vol. 2, No. 4, (2012).

R. A. Fisher. The Design of Experiments, Oliver and Boyd, Edinburgh, (1935).

R. Leardi, Application of genetic algorithm–PLS for feature selection in spectral data sets, Journal Of Chemometrics; vol.14, pp. 643–655, (2000).

Rechenberg I., Cybernetic solution path of an experimental problem. Royal Aircraft Establishment, Farnborough p. Library Translation 1122, (1965).

Rechenberg I., Evolutions Strategy: Optimization technique System and Principle in Biological Evolution. Dr.-Ing. Thesis, Technical University of Berlin, Department of Process Engineering, (1971).

Safarzynska, K., and J. Bergh, Evolutionary models in economics: A survey of methods and building blocks. Journal of Evolutionary Economics, (2009)

S. Hartmann, A Competitive Genetic Algorithm for Resource-Constrained
Project Scheduling, (1998).

S. L. K. Pond, D. Posada, M. B. Gravenor, C. H. Woelk and S. D.W. Frost, GARD: a genetic algorithm for recombination detection, Bioinformatics Applications Note, Vol. 22 no. 24 , pp 3096–3098, (2006).

S. N. Sivanandam, S. N. Deepa, Introduction to Genetic Algorithms, ISBN 978-3-540-73189-4 Springer Berlin Heidelberg New York, Springer-Verlag Berlin Heidelberg , pp. 39-71,(2008).

Schwefel H-P., Kybernetische Evolution als Strategie der exprimentellen Forschung in der Strömungstechnik. Master’s thesis, Technical University of Berlin, Germany, (1965).

Shimamoto, N., Hiramatsu, A. and Yamasaki, K. A dynamic routing control based on a genetic algorithm, IEEE International Conference on Neural Networks, 1993, vol.2, pp. 1123-1128, (2000).

Spears, W., Recombination parameters, in: *The Handbook of Evolutionary Computation,* T. B¨ack, D. B. Fogel and Z. Michalewicz, eds, Chapter E1.3, IOP Publishing and Oxford University Press, Philadelphia, PA, pp. E1.3:1–E1.3:13, (1997).

Spears, W. M. and De Jong, K. A., On the virtues of parameterized uniform crossover, in: *Proc. 4th Int. Conf. on Genetic Algorithms,* (1994).

Syswerda, G., Uniform crossover in genetic algorithms, in: *Proc. 3rd Int. Conf. on Genetic Algorithms,* pp. 2–9,(1989).

Tsang, E., P. Lsasi, and D. Quintana. A special session on evolutionary computation in finance and economics. The annual congress on evolutionary computation, Norway, (2009).

Timo Eloranta and Erkki Makinen, TimGa: A genetic algorithm for drawing Undirected Graphs. Divulgacioncs Mathematics, vol.9 pp. 155-171, (2001).

T. Miquelez, E. Bengoetxea, P. Larranaga, Evolutionary Computation Based On Bayesian Classifiers, Int. J. Appl. Math. Comput. Sci., Vol. 14, No. 3, pp 335–349, (2004).

Tank K. S., Man K. F., Kwong S and He Q. Genetic algorithms and their applications, IEEE Signal Processing Magazine, 13(6), pp. 22-37, (1996).

T. Back and Frank Hoffmeister. Adaptive search by evolutionary algorithms. In W. Ebeling, M. Peschel, and W. Weidlich, editors, *Models of Selforganization in Complex Systems (MOSES)*, Akademie–Verlag, Berlin, pp. 156–163, (1992).

Tanese, Reiko. Distributed Genetic Algorithm for Function Optimization. PhD. dissertation. Department of Electrical Engineering and Computer Science. University of Michigan. (1989).

T. Back and F.Hoffmeister, “Extended Selection Mechanisms in Genetic Algorithms”,ICGA4, pp. 92-99, (1991).