Genetic Algorithm

Tom V. Mathew
Assistant Professor,
Department of Civil Engineering,
Indian Institute of Technology Bombay, Mumbai-400076.

<vmtom@civil.iitb.ac.in>

1 Introduction

Genetic Algorithms are a family of computational models inspired by 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 as as to preserve critical information. 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. One then evaluates these structures and allocated reproductive opportunities in such a way that these chromosomes which represent a better solution to the target problem are given more chances to 'reproduce' than those chromosomes which are poorer solutions. The 'goodness' of a solution is typically defined with respect to the current population.

1.1 Background

Many human inventions were inspired by nature. Artificial neural networks is one example. Another example is Genetic Algorithms (GA). GAs search by simulating evolution, starting from an initial set of solutions or hypotheses, and generating successive "generations" of solutions. This particular branch of AI was inspired by the way living things evolved into more successful organisms in nature. The main idea is survival of the fittest, a.k.a. natural selection.

A chromosome is a long, complicated thread of DNA (deoxyribonucleic acid). Hereditary factors that determine particular traits of an individual are strung along the length of these chromosomes, like beads on a necklace. Each trait is coded by some combination of DNA (there are four bases, A (Adenine), C (Cytosine), T (Thymine) and G (Guanine). Like an alphabet in a language, meaningful combinations of the bases produce specific instructions to the cell.

Changes occur during reproduction. The chromosomes from the parents exchange randomly by a process called crossover. Therefore, the offspring exhibit some traits of the father and some traits of the mother. A rarer process called mutation also changes some traits. Sometimes an error may occur during copying of chromosomes (mitosis). The parent cell may have -A-C-G-C-T- but an accident may occur and changes the new cell to -A-C-T-C-T-. Much like a typist copying a book, sometimes a few mistakes are made. Usually this results in a nonsensical word and the cell does not survive. But over millions of years, sometimes the accidental mistake produces a more beautiful phrase for the book, thus producing a better species.

1.2 Natural Selection

In nature, the individual that has better survival traits will survive for a longer period of time. This in turn provides it a better chance to produce offspring with its genetic material. Therefore, after a long period of time, the entire population will consist of lots of genes from the superior individuals and less from the inferior individuals. In a sense, the fittest survived and the unfit died out. This force of nature is called natural selection.

The existence of competition among individuals of a species was recognized certainly before Darwin. The mistake made by the older theorists (like Lamarck) was that the environment had an effect on an individual. That is, the environment will force an individual to adapt to it. The molecular explanation of evolution proves that this is biologically impossible. The species does not adapt to the environment, rather, only the fittest survive.

1.3 Simulated Evolution

To simulate the process of natural selection in a computer, we need to define the following: A representation of an individual At each point during the search process we maintain a "generation" of "individuals." Each individual is a data structure representing the "genetic structure" of a possible solution or hypothesis. Like a chromosome, the genetic structure of an individual is described using a fixed, finite alphabet. In GAs, the alphabet 0, 1 is usually used. This string is interpreted as a solution to the problem we are trying to solve.

For example, say we want to find the optimal quantity of the three major ingredients in a recipe (say, sugar, wine, and sesame oil). We can use the alphabet 1, 2, 3 ..., 9 denoting the number of ounces of each ingredient. Some possible solutions are 1-1-1, 2-1-4, and 3-3-1.

As another example, the traveling salesperson problem is the problem of finding the optimal path to traverse, say, 10 cities. The salesperson may start in any city. A solution is a permutation of the 10 cities: 1-4-2-3-6-7-9-8-5-10.

As another example, say we want to represent a rule-based system. Given a rule such as "If color=red and size=small and shape=round then object=apple" we can describe it as a bit string by first assuming each of the attributes can take on a fixed set of possible values. Say color=red, green, blue, size=small, big, shape=square, round, and fruit=orange, apple, banana, pear. Then we could represent the value for each attribute as a sub-string of length equal to the number of possible values of that attribute. For example, color=red could be represented by 100, color=green by 010, and color=blue by 001. Note also that we can represent color=red or blue by 101, and any color (i.e., a "don't care") by 111. Doing this for each attribute, the above rule might then look like: 100 10 01 0100. A set of rules is then represented by concatenating together each rule's 11-bit string. For another example see page 620 in the textbook for a bit-string representation of a logical conjunction.

1.4 Genetic algorithm vocabulary

Explanation of Genetic Algorithm terms:

Genetic Algorithms	Explanation
Chromosome(string, individual)	Solution (coding)
Genes (bits)	Part of solution
Locus	Position of gene
Alleles	Values of gene
Phenotype	Decoded solution
Genotype	Encoded solution

2 The Canonical Genetic Algorithm

2.1 Concepts

Genetic Algorithms are search algorithms that are based on concepts of natural selection and natural genetics.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). The genetic algorithm differs 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. It also uses objective function information without any gradient information. 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.

2.2 Basic Principle

The working principle of a canonical GA is illustrated in Fig. 1. 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. They are described in detail in the following subsection.

```
/*Algorithm GA */
formulate initial population
randomly initialize population
repeat
    evaluate objective function
    find fitness function
    apply genetic operators
        reproduction
        crossover
        mutation
until stopping criteria
```

Figure 1: The Working Principle of a Simple Genetic Algorithm

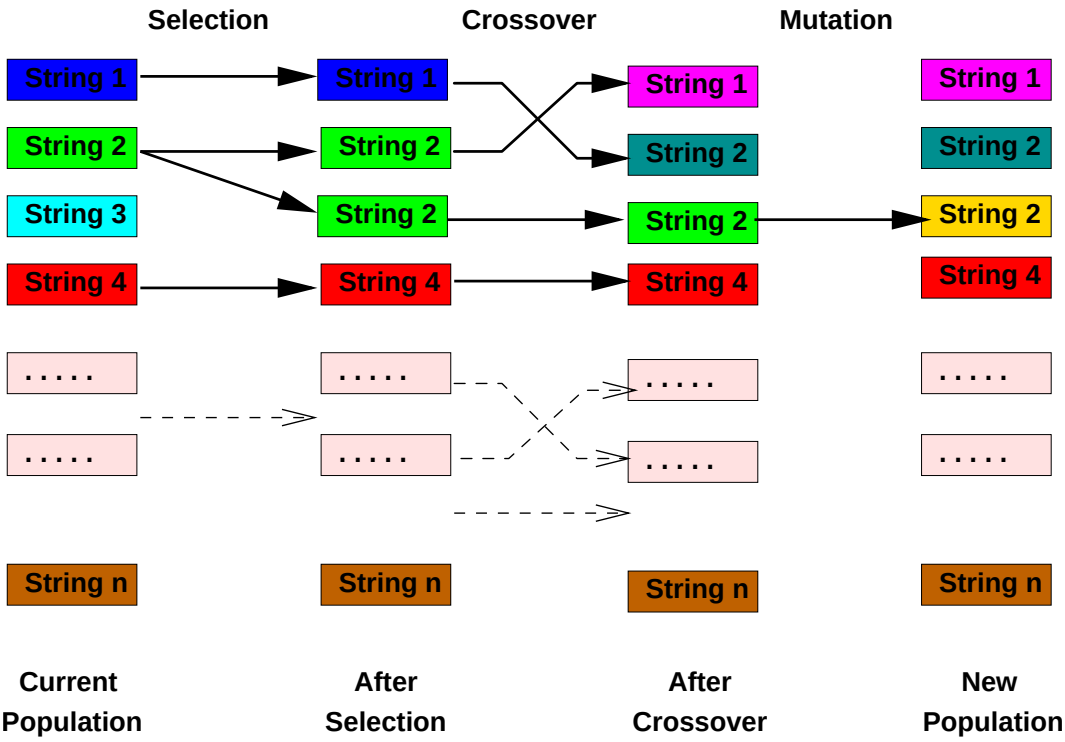


Figure 2: *The basic GA operations: One generation is broken down into a selection phase and recombination phase. Strings are assigned into adjacent slots during selection.*

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. Hence, in the first step a population having P individuals is generated by pseudo random generators whose individuals represent a feasible solution. This is a representation of solution vector in a solution space and is called initial solution. This ensures the search to be robust and unbiased, as it starts from wide range of points in the solution space.

In the next step, individual members of the population are evaluated to find the objective function value. In this step, the exterior penalty function method is utilized to transform a constrained optimization problem to an unconstrained one. This is exclusively problem specific. In the third step, the objective function is mapped into a fitness function that computes a fitness value for each member of the population. This is followed by the application of GA operators.

2.3 Working Principle

To illustrate the working principles of GAs, an unconstrained optimization problem is considered. Let us consider following maximization problem,

$$\text{M}aximize\ f(x), \quad x_i^l \leq x_i \leq x_i^u, \quad i = 1, 2, ..., N, \tag{1}$$

where, x_i^l and x_i^u are the lower and upper bound the variable x_i can take. Although a maximization problem is considered here, a maximization problem can also be handled using GAs. The working of GAs is completed by performing the following tasks.

2.4 Coding

In order to use GAs to solve the above problem (equation 1), variables x_i's are first coded in some string structures. It is important to mention here that the coding of the variables is not absolutely necessary. There exist some studies where GAs are directly used on the variables themselves, but here these exceptions are ignored and the working principles of a simple genetic algorithm is discussed.

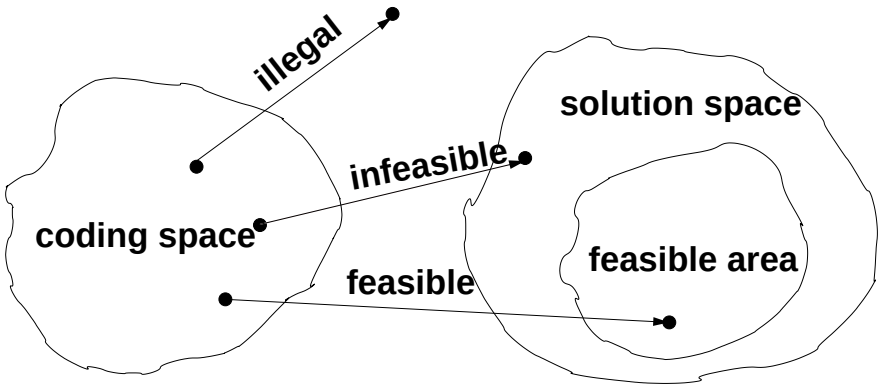


Figure 3: Coding in GA

Binary-coded strings having 1's and 0's are mostly used. The length of the string is usually determined according to the desired solution accuracy. For example, if four bits are used to code each variable in a two-variable optimization problem, the strings (0000 0000) and (1111 1111) would represent the points $(x_1^l, x_2^l)^T$ $(x_1^u, x_2^u)^T$ respectively, because the sub-strings (0000) and (1111) have the minimum and the maximum decoded values. Any other eight bit string can be found to represent a point in the search space according to a fixed mapping rule. Usually, the following linear mapping rule is used:

$$x_i = x_i^l + \frac{x_i^u - x_i^l}{2^\beta - 1} \sum_{j=0}^{\beta} \gamma_j 2^j \tag{2}$$

In the above equation, the variable x_i is coded with sub-string s_i of length β. The decoded value of a binary sub-string s_i is calculated as $\sum_{j=0}^{\beta} \gamma_j 2^j$ where $s_i \in (0, 1)$ and the string s is represented as

$(s_{\beta-1}s_{\beta-2}....s_2s_1s_0)$. For example, a four bit string (0111) has a decoded value equal to $((1)2^0 + (1)2^1 + (1)2^2 + (0)2^3)$ or 7. It is worthwhile to mention here that with four bits to code each variable, there are only 2^4 or 16 distinct sub-strings possible because each bit-position can take a value either 0 to 1. The accuracy that can be obtained with a four bit coding is only approximately $1/16^{th}$ of the search space. But as this string length is increased by 1, the obtainable accuracy increases exponentially to $1/32^{th}$ of the search space. It is not necessary to code all variables in equal sub-string lengths. The length of a sub-string representing a variable depends on the desired accuracy in that variable. The length of the sub-string varies with the desired precision of the results, the longer the string length, the more the accuracy. The relationship between string length β and precision α is

$$(x_i^u - x_i^l)10^\alpha \leq (2^\beta - 1). \tag{3}$$

Once the coding of the variables is complete, the corresponding point $x = (x_1, x_2, ..., x_N)^T$ can be found. (Eq. 2). There after, the function value at the point x can also be calculated by substituting x in the given objective function $f(x)$.

2.5 Fitness Function

As mentioned earlier, GAs mimic the survival-of-the-fittest principle of nature to make a search process. Therefore, GAs are naturally suitable for solving maximization problems. Maximization problems are usually transformed into maximization problem by suitable transformation. In general, a fitness function $F(i)$ is first derived from the objective function and used in successive genetic operations. Fitness in biological sense is a quality value which is a measure of the reproductive efficiency of chromosomes. 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. Certain genetic operators require that the fitness function be non-negative, although certain operators need not have this requirement. For maximization problems, the fitness function can be considered to be the same as the objective function or $F(i) = O(i)$. For minimization problems, to generate non-negative values in all the cases and to reflect the relative fitness of individual string, it is necessary to map the underlying natural objective function to fitness function form. A number of such transformations is possible. Two commonly adopted fitness mappings is presented below.

$$\mathcal{F}(x) = \frac{1}{1 + f(x)} \tag{4}$$

This transformation does not alter the location of the minimum, but converts a minimization problem to an equivalent maximization problem. An alternate function to transform the objective function to get the fitness value $F(i)$ as below.

$$\mathcal{F}(i) = V - \frac{O(i)P}{\sum_{i=1}^{P} O(i)}, \tag{5}$$

where, $O(i)$ is the objective function value of $i\,th$ individual, P is the population size and V is a large value to ensure non-negative fitness values. The value of V adopted in this work is the maximum value of the second term of equation 5 so that the fitness value corresponding to maximum value of the objective function is zero. This transformation also does not alter the location of the solution, but converts a minimization problem to an equivalent maximization problem. The fitness function value of a string is known as the string fitness.

2.6 GA operators

The operation of GAs begins with a population of a random strings 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 in the following steps.

2.7 Reproduction

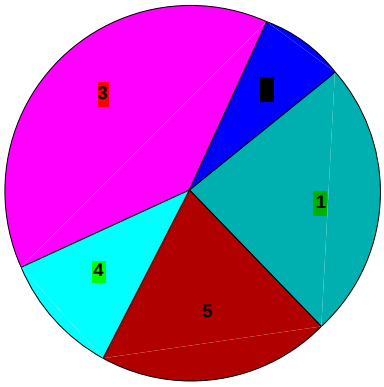
Reproduction (or selection) is an operator that makes more copies of better strings in a new population. Reproduction is usually the first operator applied on a population. Reproduction selects good strings in a population and forms a mating pool. This is one of the reason for the reproduction operation to be sometimes known as the selection operator. Thus, in reproduction operation the process of natural selection cause 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. 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.

Roulette-Wheel Selection: The commonly-used reproduction operator is the proportionate reproduction operator where a string is selected for the mating pool with a probability proportional to its fitness. Thus, the ith string in the population is selected with a probability proportional to Fi. Since the population size is usually kept fixed in a simple GA, the sum of the probability of each string being selected for the mating pools must be one. Therefore, the probability for selecting the i^{th} string is

$$p_i = \frac{\mathcal{F}_i}{\sum_{i=1}^{n} \mathcal{F}_i} \tag{6}$$

where n is the population size. One way to implement this selection scheme is to imagine a roulette-wheel with it's circumference marked for each string proportionate to the string's fitness. The roulette-wheel is spun n times. each time selecting an instance of the string chosen by the roulette-wheel pointer. Since the circumference of the wheel is marked according to a string's fitness, this roulette-wheel mechanism is expected to make $\mathcal{F}_i/\overline{\mathcal{F}}$ copies of the ith string in the mating pool. The average fitness of the population is calculated as

$$\overline{\mathcal{F}} = \sum_{i=1}^{n} \mathcal{F}_i \tag{7}$$



Population	Fitness
1	25.0
2	5.0
3	40.0
4	10.0
5	20.0

Figure 4: *A roulette-wheel marked for five individuals according to their fitness values. Third individual has a higher probability of selection than any other*

The figure shows a roulette-wheel for each individuals having different fitness values. since the third individual has higher fitness value than any other, it is expected that the roulette-wheel selection will choose the third individual more than any other individual. This roulette-wheel selection scheme can be simulated easily. Using the fitness value Fi of all strings, the probability os selecting s string pi can be calculated. Thereafter, the cumulative probability Pi of each string being copied can be calculated

by by adding the individual probabilities from the top of the list. Thus, the bottom-most string in the population should have a cumulative probability values from Pi-1 to P. The first string represents the cumulative values from zero to P1. Thus, the cumulative probability of any sting lies between 0 to 1. In order to choose n strings, n random numbers between zero to one are created at random. Thus, s string that represents the chosen random number in the cumulative probability range (calculated from the fitness values) for the string is chosen to the mating pol. This way the string with a higher fitness value will represent a larger range in the cumulative probability values and therefore has a higher of being copied into the mating pool. On the other hand, a string with a smaller fitness value represents a smaller range in cumulative probability values and has a smaller probability of being copied into the mating pool.

Stochastic remainder selection: A better selection scheme is also presented here The basic idea of this selection is to remove or copy the strings depending on the values of reproduction counts. This is achieved by computing the reproduction count associated with each string. Reproduction count is computed based on the fitness value by stochastic remainder selection without replacement as it is superior to other schemes.Hence, this scheme is recommended. First the probability of selection p_s is calculated as $p_s = F(i)/\sum F(i)$. The expected number of individuals of each string is calculated as $ei = p_s \times P$, where P is the population size. The fractional parts of ei are treated as probabilities with which individuals are selected for reproduction. One by one Bernoulli trials (i.e. weighted coin tosses) are performed using the fractional part of ei. For example, a string with $ei = 1.5$ will get a single count surely and another with a probability of 0.5. This is continued till all the candidates in the population are examined. Reproduction is done based on this computed reproduction count. Individuals with 0 count are eliminated from the population. Other individuals with non-zero counts get multiple copies in population equal to the value of their counts. The size of the population is kept constant and this completes the reproduction operation. Different selection schemes vary in principle by assigning different number of copies to better strings in the population but in all selection schemes the essential idea is that more copies are allocated to the strings with higher fitness values.

2.8 Crossover

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. 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. When a crossover probability, defined here as p_c is used, only $100p_c$ per cent strings in the population are used in the crossover operation and $100(1 - p_c)$ per cent of the population remains as they are in the current population. A crossover operator is mainly responsible for the search of new strings even though mutation operator is also used for this purpose sparingly.

Many crossover operators exist in the GA literature.One site crossover and two site crossover are the most common ones adopted. In most crossover operators, two strings are picked from the mating pool at random and some portion of the strings are exchanged between the strings. Crossover operation is done at string level by randomly selecting two strings for crossover operations. A one site crossover operator is performed by randomly choosing a crossing site along the string and by exchanging all bits on the right side of the crossing site as shown in Fig. 5.

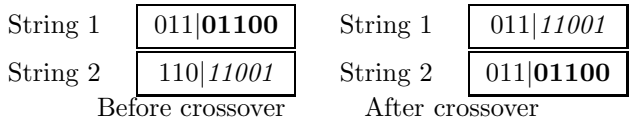


Figure 5: One site crossover operation

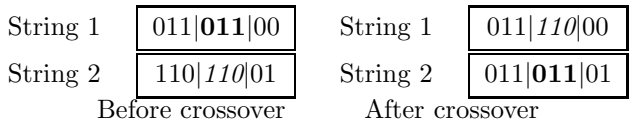


Figure 6: Two site crossover operation

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 are exchanged to form a new pair of strings. The new strings are thus a combination of the old strings. Two site crossover is a variation of the one site crossover, except that two crossover sites are chosen and the bits between the sites are exchanged as shown in Fig. 6.

One site crossover is more suitable when string length is small while two site crossover is suitable for large strings. Hence the present work adopts a two site crossover. The underlying objective of crossover is to exchange information between strings to get a string that is possibly better than the parents.

2.9 Mutation

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 p_m. 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 neither reproduction nor crossover operator described above will be able to create 1 in that position. The inclusion of mutation introduces probability p_m of turning 0 into 1.

These three operators are simple and straightforward. The reproduction operator selects good strings and 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.

3 GA - Illustrated

To understand the working of GA, a simple two variable function is solved using GA. Detailed steps and solution obtained are listed below. Consider the following minimization problem

$$f(x_1, x_2) = ({x_1}^2 + x_2 - 11) + (x_1 + {x_2}^2 - 7)^2 \tag{8}$$

in the interval $0 \leq x_1,\ x_2 \leq 6$. The true solution to this problem is $(3, 2)^T$ having a function value equal to zero.

Step 1: To solve this problem using genetic algorithm, a binary coding is chosen to represent variables x_1 and x_2. In the calculation here, 10-bits are chosen for each variable, thereby making the total string length equal to 20. With 10 bits, we can get a solution accuracy of (6-0)/(2^{10}-1) or 0.006 in the interval (0,6). The crossover and mutation probabilities are assigned to be 0.8 and 0.05 respectively. The population size is 20 and the number of generation is 30. The built in c random number generator is used and stochastic sampling with out replacement is used for selection The next step is to evaluate each string in the population We calculate the fitness of the first string.

Step 2: The next step is to calculate the fitness of each population. This is done by decoding the strings. The first sub-string (1000001110) decodes to a value equal to (2^9+2^3+2^2+2^1) or 525. Thus the corresponding parameter is 0 + (6-0) * 525/1023 or 3.09 Let a second sub-string (0001110000) decodes to a value equal to 112. Thus the corresponding parameter is 0 + (6-0) * 112/1023 or 0.66. so the first string corresponds to the point $x^{(0)} = (3.09, 0.66)^T$ Substituting these values in to the objective function we will the function value, which is equal to 12.816. Since the problem is minimization one, the fitness function of this point is calculated as $\mathcal{F}(x^{(1)}) = 1.0/(1.0+12.816) = 0.072$ This value is used in the reproduction operation. The table (3) provide the details of other individuals.

Step 3: Since the iteration has not reached the upper limit, we proceed to next step.

Step 4: In this step the selection is done using stochastic sampling without replacement The will generate reproduction count based on the fitness value. For example the first individual has a count 0, the second has 1 and the fourth has 2. Note that first has a very high objective function value and hence got zero count. This individual will not be copied. But the fourth one with a low objective function value gets count 2 and will get two copies in the next population. This completes the selection or reproduction. This is followed by the application of crossover and mutation operation and the resulting new population is show in the later half of the table. Note that in the due to these operation, individual with high objective function value are eliminated. However, due to crossover and mutation some individual solutions got worse (12,17) while some other got better (6,11) The improvement is not significant in the first generation. But comparing the initial and solution at 30, one can infer that all the individuals actually improved. See figure (7 and 8)

4 GAs and Traditional Methods

As seen from the above description of the working principles of GAs, they are radically different from most of the traditional optimization methods. However, the fundamental differences are described subsequently. GAs work with a string-coding of variables instead of the variables. The advantage of working with a coding of variables is that the coding discretizes the search space, even though the function may be

No	String 1	String 2	x_1	x_2	z	f	c	String 1	String 2	x_1	x_2	x	f
0	1000001110	0001110000	3.0	0.6	12.8	.072	0	010000100	01000011011	1.5	3.1	50.2	.020
1	0001011101	1100000100	0.5	4.5	235.4	.004	1	000101110	10010110000	0.5	1.0	122.5	.008
2	0010110110	0000110000	1.0	0.2	126.0	.008	1	001011011	11001110000	1.0	3.6	94.0	.011
3	0100001001	1000011011	1.5	3.1	50.0	.020	2	010000100	10000011001	1.5	0.1	100.6	.010
4	1011001101	0100111010	4.2	1.8	73.0	.014	1	101000111	11010111110	3.8	4.1	252.2	.004
5	1001111001	0111100011	3.7	2.8	53.9	.018	1	100111101	10111100011	3.7	2.8	55.0	.018
6	1000111011	1101111111	3.3	5.2	601.2	.002	0	001110101	11001000011	1.3	3.4	67.4	.015
7	0011100111	1001110100	1.3	3.6	92.7	.011	1	000110111	00010110100	0.6	1.0	118.2	.008
8	0100101011	0010010010	1.7	0.8	70.3	.014	1	010010100	10010010010	1.7	0.8	71.0	.014
9	0010101011	1000110000	1.0	3.2	67.9	.014	1	001110111	00101010000	1.4	1.9	53.0	.018
10	0100101001	0011110000	1.7	1.4	53.7	.018	1	010010100	11100000100	1.7	4.5	244.1	.004
11	1100110000	0101111110	4.7	2.2	207.9	.005	1	011010000	00001111110	2.4	0.7	34.6	.028
12	0110001101	1011111111	2.3	4.5	243.4	.004	1	111000110	10110111101	5.3	2.6	427.7	.002
13	0000100111	1011011000	0.2	4.2	175.9	.006	1	000001010	10100100110	0.1	1.7	100.9	.010
14	0010111001	0111011001	1.0	2.7	52.8	.019	1	000011100	10011011001	0.3	1.2	117.9	.008
15	0011101110	1001000011	1.4	3.4	67.1	.015	2	001000101	11000100011	0.8	3.2	67.6	.015
16	1111110000	0100001110	5.9	1.5	654.0	.002	0	110001010	10110101110	4.6	2.5	183.2	.005
17	0110010101	0110101110	2.3	2.5	11.0	.083	2	011010011	11111011000	2.4	5.7	829.6	.001
18	0110001110	0011000101	2.3	1.1	30.4	.032	2	011000111	00011000101	2.3	1.1	30.4	.032
19	1100111110	1101001111	4.8	4.9	820.4	.001	0	011000111	00011010111	2.3	1.2	27.8	.035

Table 1: Evaluation and Reproduction Phases on a Population

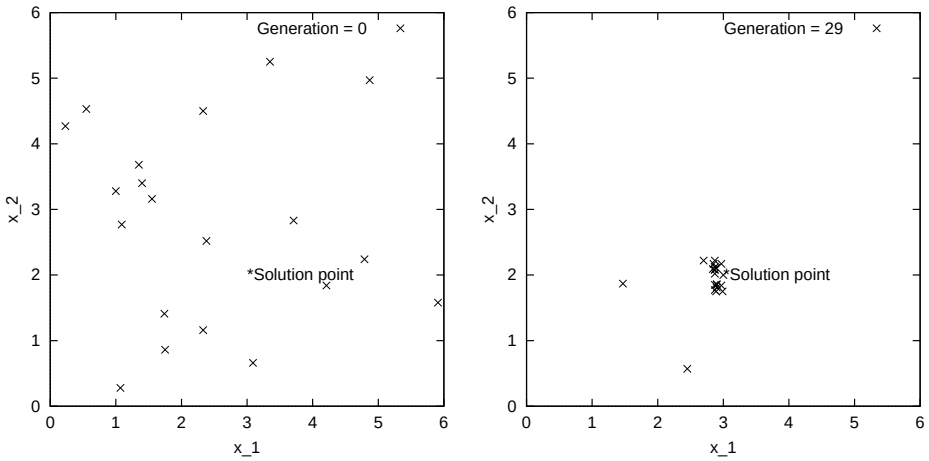


Figure 7: The objective function value of the best point in the population for GA runs with different random seeds. All the three cases converge quickly to the optimum point

continuous. On the other hand, since GAs require only function values at various discrete points, a discrete or discontinuous function can be handled with no extra burden. This allows GAs to be applied to a wide variety of problems. Another advantage is that the GA operators exploit the similarities in string-structures to make an effective search. The most striking difference between GAs and many traditional optimization methods is that GAs work with a population of points instead of a single point. Because there are more than one string being processed simultaneously, it is very likely that the expected GA solution may be a global solution. Even though some traditional algorithms are population based, like Box's evolutionary optimization and complex search methods, those methods do not use previously obtained information efficiently. In GAs, previously found good information is emphasized using reproduction operator and propagated adaptively through crossover and mutation operators. Another advantage with a population-based search algorithm is that multiple optimal solutions can be captured in the population easily, thereby reducing the effort to use the same algorithm many times.

Genetic algorithms differ from conventional optimization and search procedures in several fundamental ways. It can be summarized as follows:

1. Genetic algorithms work with a coding of solution set, not the solutions themselves.
2. Genetic algorithms search from a population of solutions, not a single solution.
3. Genetic algorithms use payoff information (fitness function), not derivatives or other auxiliary knowledge.
4. Genetic algorithms use probabilistic transition rules, not deterministic rules.

4.1 Exploitation and exploration

Search is one of the more universal problem-solving methods for such problems where one cannot determine a priory the sequence of steps leading to a solution. Search can be performed with either blind strategies or heuristic strategies. Blind search strategies do not use information about the problem domain. Heuristic search strategies use additional information to guide the search along with the best search directions. There are two important issues in search strategies: exploiting the best solution and

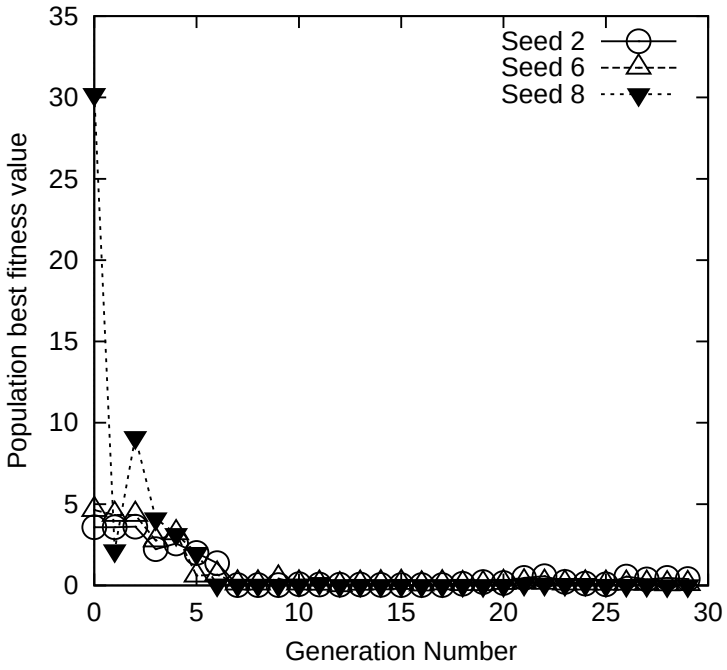


Figure 8: The objective function value of the best point in the population for GA runs with different random seeds.

exploring the search space. Hill-climbing is an example of a strategy which exploits the best solution for possible improvement while ignoring the exploration of the search space. Random search is an example of a strategy which explores the search space while ignoring the exploitation of the promising regions of the search space. Genetic algorithms are a class of general-purpose can make a remarkable balance between exploration and exploitation of the search space. At the beginning of genetic search, there is a widely random and diverse population and crossover operator tends to perform widespread search for exploring all solution space. As the high fitness solutions develop, the crossover operator provides exploration in the neighborhood of each of them. In other words, what kinds of searches (exploitation or exploration) a crossover performs would be determined by the environment of the genetic system (the diversity of population), but not by the operator itself. In addition, simple genetic operators are designed as general-purpose search methods (the domain-independent search methods); they perform essentially a blind search and could not guarantee to yield an improved offspring.

4.2 Population-based search

Generally, shown in Figure 1, the algorithm for solving optimization problem is a sequence of computational steps which asymptotically converge to optimal solution. Most classical optimization methods generate a deterministic sequence of computation based on the gradient or higher-order derivatives of objective function. This point-to-point approach takes the danger of falling in local optima. Genetic algorithms perform a multiple directional search by maintaining a population of potential solutions. The population-to-population approach attempts to make the search escape from local optima. Population undergoes a simulated evolution: At each generation the relatively good solutions are reproduced while the relatively bad solutions die. Genetic algorithms use probabilistic transition rules to select someone to be reproduced and someone to die so as to guide their search toward regions of the search space with like improvement.

5 Applications of GA

Nearly everyone can gain benefits from Genetic Algorithms, once he can encode solutions of a given problem to chromosomes in GA, and compare the relative performance (fitness) of solutions. An effective

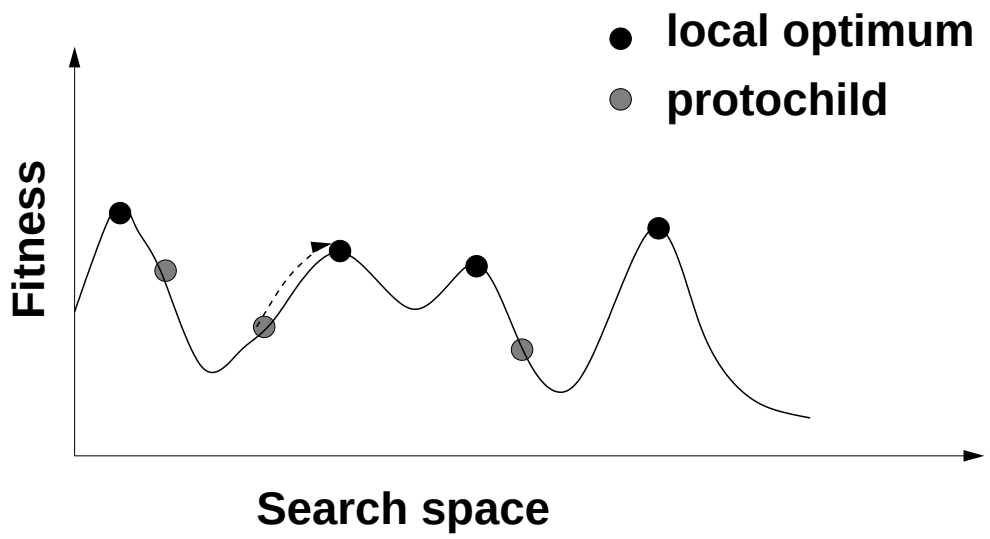


Figure 9: Transition in GA

GA representation and meaningful fitness evaluation are the keys of the success in GA applications. The appeal of GAs comes from their simplicity and elegance as robust search algorithms as well as from their power to discover good solutions rapidly for difficult high-dimensional problems. GAs are useful and efficient when

1. The search space is large, complex or poorly understood
2. Domain knowledge is scarce or expert knowledge is difficult to encode to narrow the search space
3. No mathematical analysis is available
4. Traditional search methods fail

The advantage of the GA approach is the ease with which it can handle arbitrary kinds of constraints and objectives; all such things can be handled as weighted components of the fitness function, making it easy to adapt the GA scheduler to the particular requirements of a very wide range of possible overall objectives.

GAs have been used for problem-solving and for modeling. Gas are applied to many scientific, engineering problems, in business and entertainment, including:

- Optimization: GAs have been used in a wide variety of optimization tasks, including numerical optimization, and combinatorial optimization problems such as traveling salesman problem (TSP), circuit design [Louis 1993] , job shop scheduling [Goldstein 1991] and video & sound quality optimization.
- Automatic Programming: GAs have been used to evolve computer programs for specific tasks, and to design other computational structures, for example, cellular automata and sorting networks.
- Machine and robot learning: GAs have been used for many machine- learning applications, including classification and prediction, and protein structure prediction. GAs have also been used to design neural networks, 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 individuals 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-evolutions, 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?" 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 [Chughtai 1995], the evolution of communication, and trail-following behavior in ants.

6 Conclusion

Genetic Algorithms are easy to apply to a wide range of problems, from optimization problems like the traveling salesperson problem, to inductive concept learning, scheduling, and layout problems. The results can be very good on some problems, and rather poor on others. If only mutation is used, the algorithm is very slow. Crossover makes the algorithm significantly faster. GA is a kind of hill-climbing search; more specifically it is very similar to a randomized beam search. As with all hill-climbing algorithms, there is a problem of local maxima. Local maxima in a genetic problem are those individuals that get stuck with a pretty good, but not optimal, fitness measure. Any small mutation gives worse fitness. Fortunately, crossover can help them get out of a local maximum. Also, mutation is a random process, so it is possible that we may have a sudden large mutation to get these individuals out of this situation. (In fact, these individuals never get out. It's their offspring that get out of local maxima.) One significant difference between GAs and hill-climbing is that, it is generally a good idea in GAs to fill the local maxima up with individuals. Overall, GAs have less problems with local maxima than back-propagation neural networks.

If the conception of a computer algorithms being based on the evolutionary of organism is surprising, the extensiveness with which this algorithms is applied in so many areas is no less than astonishing. These applications, be they commercial, educational and scientific, are increasingly dependent on this algorithms, the Genetic Algorithms. Its usefulness and gracefulness of solving problems has made it the a more favorite choice among the traditional methods, namely gradient search, random search and others. GAs are very helpful when the developer does not have precise domain expertise, because GAs possess the ability to explore and learn from their domain.

In this report, we have placed more emphasis in explaining the use of GAs in many areas of engineering and commerce. We believe that, through working out these interesting examples, one could grasp the idea of GAs with greater ease. We have also discuss the uncertainties about whether computer generated life could exist as real life form. The discussion is far from conclusive and ,whether artificial life will become real life, will remain to be seen.

In future, we would witness some developments of variants of GAs to tailor for some very specific tasks. This might defy the very principle of GAs that it is ignorant of the problem domain when used to solve problem. But we would realize that this practice could make GAs even more powerful.

References

Beasley, D., Bull, D. R. and Martin, R. R. (1993), "An overview of genetic algorithms: Part 2, research topics", *University Computing* , Vol. 15, pp. 170–181. citeseer.nj.nec.com/article/beasley93overview.html.

Chambers, L. (1995), *Practical handbook of genetic algorithms:Applications, Vol.I*, CRC Press, Boca Raton, Florida.

Charbonneau, P. (1998), 'An introduction to genetic algorithms for numerical optimization'. http://www.hao.ucar.edu/public.research/si/pikaia/tutorial.html.

Deb, K. (1998), *Optimization for engineering design: Algorithms and Examples*, Prentice Hall, India.

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

Genetic algorithms : principles and perspectives: a guide to GA theory (2002), Kluwer Academic, Boston.

Goldberg, D. E. (1989), *Genetic Algorithms in search, optimization and machine learning*, Addison-Wesley, Massachusetts.

Holland, J. (n.d.), *Adaptation in Natural and Artificial Systems*, 2nd ed., MIT Press, MIT, Cambridge.

Vose, M. D. (1999), *Simple genetic algorithm : foundations and theory*, MIT Press.

Whitley, D. (2001), 'A genetic algorithm tutorial'. http://samizdat.mines.edu/ga_tutorial.