The optimal basics for GAs

© Photo Disc

Harinath Babu Kamepalli

The Genetic Algorithm (GA) is becoming a flagship among various techniques of machine learning and function optimization. In simple terms, an algorithm is a set of sequential steps requiring execution to achieve a task. A GA is an algorithm with principles of genetics included in it. The genetic principles "Natural Selection" and "Evolution Theory" are main guiding principles in implementing a genetic algorithm. The GA combines the adaptive nature of natural genetics and search through a randomized information exchange.

There is a multitude of search techniques. Among them calculus-based, enumerative, and random search techniques are mostly used. Calculus-based and enumerative techniques can arrive at reasonably good solutions for search spaces of smaller sizes. But when confronted with search spaces of enormous size and wide variation from point to point in their precinct, their efficacy in delivering solutions is drastically low. They are insufficiently robust for complex problems involving huge search space. This is due to their lack of ability to overcome the local optimum points and reach the global optimum point.

To overcome these local optimum points, we use a *random search* technique. Keep in mind that this randomized search is not a directionless search. The search is carried out randomly; however, the information gained from the search is utilized in guiding the next search. Genetic algorithms are an example of such a search technique.

Genetic algorithms surpass all the aforementioned limitations of conventional algorithms by using different basic building blocks. They are in the following aspects.

- GA works with a coding of the parameter set, and not the parameters themselves.
- GA searches from a population of points, and not from a single point like conventional algorithms.
- GA uses objective function information, and not derivative or other auxiliary data.
- GA use probabilistic transition rules by stochastic operands, and not deterministic rules.

The first step of a GA is randomly selecting initial search points from the total search space. Each and every point in the search space corresponds to one set of values for the problem's parameters. Each parameter is coded with a string of bits. The individual bit is called a "gene." The content of each gene is called an "allele." The total string of such genes for all parameters in a written sequence is called a "chromosome." So there exists a chromosome for each point in the search space.

The set of search points selected and used for processing is called a "population"; i.e. a population is a set of chromosomes. The number of chromosomes in a population is called the "population size." The total number of genes in a string is called the "string length."

The population is processed and evaluated through various operators of the GA to generate a new population. This process is carried out until a global optimum point is reached. The two parts of this process are called "Generation" and "Evaluation."

During evaluation, we define a fitness function and evaluate the fitness for each chromosome of the population. This fitness indicates the suitability of the values of the parameters—as represented by that chromosome—to create an optimal solution for the problem. This fitness is used as a bias for selecting the parents and generating a new population from the existing one.

Thus, we are testing a significant number of solutions simultaneously since each chromosome represents a solution. This is referred to as "Implicit Parallelism." GA is the only search technique that employs implicit parallelism.

The three phases

A Genetic Algorithm includes: a string representation of points in a search space; a set of genetic operators for generating new search points; a fitness function to evaluate the search points and a stochastic assignment to control the genetic operations. Simplicity of operation and power of effect are two main attractions of the GA approach. The approach typically has three phases:

1. Initialization
2. Evaluation
3. Genetic Operation

Initialization

Initialization is the generation of the

0278-6648/01/$10.00 © 2001 IEEE

initial population of chromosomes; i.e., the initial search points. Two parameters—population size and string length—need to be judiciously selected before this job is performed.

The population size is a direct indication of how effective the representation is for the whole search space. The population size affects both the ultimate performance and the efficiency of the GA. If too small, the chance that the chromosomes in the population cover the entire search space is low. This makes it difficult to obtain the global optimum solution and leads to a local optimum solution.

Please note that this stagnation at the local optimum is a result of premature convergence. Thus, a large population size is preferable to avoid this premature convergence and to reach a global optimum point. But a population size that is too large decreases the rate of convergence. In the worst case scenario, it may lead to divergence. Hence, the population size needs to be selected based on the size of the search space.

Deciding the string's length depends on the accuracy requirements of the optimization problem. The higher the string length, the higher will be the resolution and accuracy. But this leads to a slow convergence. Also, the number of parameters in the optimization problem will directly effect the number of bits in a chromosome, i.e. string length. Keeping all the above constraints in mind, the string length is chosen appropriately.

After selecting string length and population size, the initial population is generated as a set of strings of bits, either "0" or "1." Random number generation techniques are used to accomplish this task. These strings of bits contain the information related to the problem's parameters in an encoded format. Any encoding technique can be used but binary encoding is convenient and the one mostly used.

Now, from the initial population, chromosomes are decoded and all parameters of the optimization problem are calculated for each chromosome. This results in a set of solutions whose size is equal to population size.

Evaluation

In the evaluation phase, the suitability of each solution from the initial set as *the* solution for the problem is determined. To do this, a function called the "fitness function" is defined. This is used as a deterministic tool to evaluate the fitness of each chromosome. The optimization problem may be a minimization or a maximization type. In either case, the fitness function is selected so that the fittest solution is the one nearest to the global optimum point. The programmer is allowed to use any fitness function that adheres to these requirements. This flexibility with the GA is one of its fortes.

Overall for a typical optimization problem, the evaluation phase involves: calculating individual parameters of the problem by decoding the encoded strings from the population; testing any equality or inequality constraints that need to be satisfied; evaluating the objective function and, finally, evaluating fitness based on the fitness function. This evaluation is discrete in nature vis-à-vis some genetic operators which operate on more than one chromosome at a time.

Genetic operation

In this phase, a new population is generated from the existing one by examining the fitness values of various chromosomes and applying the genetic operators. These genetic operators are *reproduction, crossover* and *mutation*. This phase is carried out if we are not satisfied with the solution obtained earlier. The survival of the fittest means transferring the highly fit chromosomes to the next generation of strings and combining different strings to explore new search points.

Reproduction

Reproduction is simply an old chromosome being copied into a mating pool based on its fitness value. Highly fit chromosomes get a higher number of copies in the next generation. Copying chromosomes according to their fitness means that the chromosomes with a higher fitness value have a higher probability of contributing one or more offspring in the next generation.

Crossover

Crossover is a recombination operation. Here the gene information contained in the two selected parents is utilized to generate two children bearing some of their parents' useful characteristics. The kids are expected to be more fit than the parents.

There are various techniques that are used for performing this crossover. But first, we need to pick up two parents from the existing population to perform the crossover. This selection can be done using the random selection or the roulette wheel selection method. In the random selection technique, the parents are picked up randomly from the existing population. In roulette wheel selection process, a little more streamlining is involved.

Basically, a linear search is usually implemented through a roulette wheel whose "slot sizes" are in proportion to the string's fitness values. This is achieved using the following steps.

1. The fitness of all the strings (fitsum) is totaled.

2. A random real number (rand-sum) between 0 and fitsum is generated.

3. Starting with the first member of the existing population, for each member "n," the fitness sum of its members "1" to "n" is compared with the randomly generated number.

4. If $\sum$(fitness of member n) is > than rand-sum, n is selected as a parent. Otherwise, the process is continued by incrementing n.

All four steps are useful in selecting a parent. So before performing a crossover, we execute these steps twice. Obviously through this roulette wheel process, we are giving more reproductive chances to the fitter population members. Thus, we are ensuring that the parent chromosomes are chosen based on their objective function values.

The convergence rates and efficiency of GA with the roulette wheel selection technique is far superior than the random selection technique. With the roulette wheel selection technique, a still faster rate of convergence can be achieved by sorting the population in descending order of "fitness" before selecting parents.

Now the crossover is carried out using any one of these three methods.

1. Simple or single point crossover
2. Multipoint crossover
3. Uniform crossover

Simple point crossover

In this method, crossover is carried out at a single point as illustrated in the following example. Let Parl and Par2 be the two parents selected for crossover. Assume the strings parl and par2 as below.

Parl: 1 1 0 0 0 1 0 1

Par2: 1 0 1 1 0 1 1 1

Now, a crossover site is selected randomly as an integer between 1 and the string length. Here this crossover site is 4. Then children Chldl and Chld2 are generated as follows.

Chld1: 1 2 3 4 5 6 7 8 = 1 1 0 0 0 1 1 1

<-Parl->|<-Par2->

Chld2: 1 2 3 4 5 6 7 8 = 1 0 1 1 0 1 0 1

<-Par2->|<-Parl->

Multipoint crossover

This method is similar to single point crossover except that more than one crossover site is randomly selected. Also, the contents of Chld1 and Chld2 are selected alternatively from Parl and Par2 by changing from one parent to the another at the crossover sites.

Uniform crossover

In this method, a crossover is performed over the entire string length of bits. First, a mask is generated randomly. This mask is nothing but a string of bits of "0" or "1" with the same length size. With the information in the mask, we generate the children as follows.

Parl: 1 1 0 0 1 0 1 1

Par2: 0 1 0 0 0 1 0 0

Mask: 0 0 1 0 1 1 0 1

Chld1: 1 1 0 0 0 1 1 0 (If mask=0, Chld1=Par1 & Chld2=Par2)

Chld2: 0 1 0 0 1 0 0 1 (If mask=1,Chld1=Par2 & Chld2=Par1)

Here we need to generate for each crossover. Thus, the number of masks needed is equal to the number of crossovers to be performed. But we don't need to store these masks. We generate them as and when required and discard them thereafter.

As we have seen, each crossover results in two children. So the number of crossovers required for the next generation depends on the number of children we need. Usually, some of the best parents are copied "as is" into the next generation with the additional required strings generated as children. This phenomenon of copying the best parents into the next generation is called "Elitism." The number of parents so copied is indicated by a parameter of GA called the "Percentage of Elitism (P_e)." This is nothing but the percentage of parents directly copied out of the total number. This elitism is basically carried out so the best strings obtained so far are not lost.

To control the number of crossovers, there also is a parameter called "Crossover Probability (P_c)." This probability is used as a decision variable before performing the crossover. A random number between "1" and "0" is generated. If that number is less than P_c, a crossover is performed. If the randomly generated number is greater than P_c, Chldl and Chld2 are directly selected as Parl and Par2. This is equivalent to when the crossover site is equal to the string length. There are various other techniques too for implementing the P_c and the programmer of GA is free to choose any one. But this technique is commonly used.

Mutation

This operator can create new genetic material in the population to maintain the population's diversity. It is nothing but a random alteration of a bit value at a particular bit position in the chromosome. The following example illustrates the mutation operation.

Original String: 1011001

Mutation site: 4 (assumption)

String after mutation: 1010001

Some programmers randomly choose whether to alternate or keep the same a particular bit in the mutation site. "Mutation probability (P_m)" is a parameter used to control the mutation. For each string, a random number between "0" and "1" is generated and compared with the P_m. If it is less than P_m, a mutation is performed on the string. Sometimes a mutation is performed bit by bit rather than by strings. This results in substantial increase in CPU time without the GA's performance increasing appreciably. So this is usually not preferred.

Thus, mutation brings in some points from the regions of search space that otherwise might not be explored. Generally, the mutation probability will be in the range of 0.001 to 0.01.

The overall process

Overall, the optimization problem that is to be solved using GA is attempted as follows.

Step 1: Necessary data of the problem is collected and entered as input.

Step 2: Phase 1 - Initialization is done for all the problems.

Step 3: Phase 2 and Phase 3 - Evaluation and Genetic Operation are performed repeatedly until the convergence is achieved.

Based on different combinations of operators and strategies, GAs are classified as three types.

1. Simple GA: The multipoint crossover and mutation are the operators used and the roulette wheel is the selection technique.

2. Refined GA: Uniform crossover and mutation are the operators and the roulette wheel is the selection technique. Strategies such as elitism and changing P_c and P_m are also implemented.

3. Crowding GA: This consists of uniform crossover and mutation, the random selection technique and strategies such as parent replacement and changing P_c and P_m are also implemented.

Remember, there is no hard and bound restrictions on what operator and strategies of GA a programmer has to use. The programmer can choose the operation and strategies in any combination according to his or her views.

Summary

Genetic algorithms were introduced by John Holland in early seventies as a special technique for function optimization. They are quite different from other more conventional optimization methods that are mainly stochastic in nature. A typical GA will have three phases; i.e., initialization, evaluation and genetic operation. In each phase, various parameters of GA need to be selected based on the nature of the optimization problem. A genetic algorithm is also classified based on the various combinations of parameters and strategies employed. However, the designer is free to develop a hybrid genetic algorithm. The main goal is to deliver the most enhanced performance possible to the optimization problem.

Read more about

• Goldberg. D.E., *Genetic Algorithms in Search, Optimization and Machine Learning,* Addison Wesley, 1989.

• Davis. L., *Handbook of Genetic Algorithms,* Van Nostrand Reinhold, '91.

• John R. Koza, *Genetic programming—On the Programming of Computers by Means of Natural Selection,* MIT press, 1992.

• Albrecht, Reeves, Steele, "Artificial Neural Networks and Genetic Algorithms," *Proceedings of International Conference in Innsbruck, Austria*, 1993.

About the author

Mr. Harinath Babu Kamepalli wrote this article in his final year as a B.Tech (Electrical and Electronics Engineering) student of Regional Engineering College, Warangal, India. Mr. Kamepalli received a medal and special ranks in A.P. State Mathematical Olympiad in the years 1996, 1994, and 1997 respectively.

After his B. Tech., Mr. Kamepalli obtained a Master of Science in EE from the University of Minnesota, Twin Cities, MN. Currently, he is working as a CAD Engineer at Sun Microsystems, Inc. Sunnyvale, CA. For additional details, you can e-mail him at <hbk@ieee.org>or visit the web site <http://www.angelfire.com/on/hbk>.