GENETIC ALGORITHMS: A PROBLEM SOLVING APPROACH

Jyoti*, Neetu Gupta

Asst. Prof - H.A.S Department, YMCA Univ. of Sci. & Tech, Faridabad

Abstract
Genetic Algorithms (GAs) are adaptive heuristic search algorithm premised on the evolutionary ideas of natural selection and genetic. The basic concept of GAs is designed to simulate processes in natural system necessary for evolution, specifically those that follow the principles first laid down by Charles Darwin of survival of the fittest. As such they represent an intelligent exploitation of a random search within a defined search space to solve a problem. First pioneered by John Holland in the 60s, Genetic Algorithms has been widely studied, experimented and applied in many fields in engineering worlds. Not only does GAs provide an alternative method to solving problem, it consistently outperforms other traditional methods in most of the problems link. Many of the real world problems involved finding optimal parameters, which might prove difficult for traditional methods but ideal for GAs. However, because of its outstanding performance in optimization, GAs has been wrongly regarded as a function optimizer. In fact, there are many ways to view genetic algorithms. In this paper we aim at providing an insight into the field of genetic algorithms and its areas of application

Keywords: Genetic Algorithms, Optimization, Application of GAs.

Introduction
Genetic algorithm is a randomized search methodology having its roots in the natural selection process. Initially the neighborhood search operators (crossover and mutation) are applied to the preliminary set of solutions to acquire generation of new solutions. Solutions are chosen randomly from the existing set of solutions where the selection probability and the solution’s objective function value are proportional to each other and eventually the aforesaid operators are applied on the chosen solutions. Genetic algorithms have aided in the successful implementation of solutions for a wide variety of combinatorial problems.

History
Computer simulations of evolution started as early as in 1954 with the work of Nils Aall Barricelli. His 1954 publication was not widely noticed. Starting in 1957, the Australian quantitative geneticist Alex Fraser published a series of papers on simulation of artificial selection of organisms with multiple loci controlling a measurable trait. From these beginnings, computer simulation of evolution by biologists became more common in the early 1960s, and the methods were described in books by Fraser and Burnell (1970) and Crosby (1973). Fraser's simulations included all of the essential elements of modern genetic algorithms. In addition, Hans-Joachim Bremermann published a series of papers in the 1960s that also adopted a population of solution to optimization problems, undergoing recombination, mutation, and selection. Bremermann's research also included the elements of modern genetic algorithms. Other noteworthy early pioneers include Richard Friedberg, George Friedman, and Michael Conrad. Many early papers are reprinted by Fogel (1998).

Although Barricelli, in work he reported in 1963, had simulated the evolution of ability to play a simple game, artificial evolution became a widely recognized optimization method as a result of the work of Ingo Rechenberg and Hans-Paul Schwefel in the 1960s and early 1970s – Rechenberg's group was able to solve complex engineering problems through evolution strategies. Another approach was the evolutionary programming technique of Lawrence J. Fogel, which was proposed for generating artificial intelligence. Evolutionary programming originally used finite state machines for predicting environments, and used variation and selection to optimize the predictive logics. Genetic algorithms in particular became popular through the work of John Holland in the early 1970s, and particularly his book Adaptation in Natural and Artificial Systems (1975). Research in GAs remained largely theoretical until the mid-1980s, when The First International Conference on Genetic Algorithms was held in Pittsburgh, Pennsylvania.[5]
What is Genetic Algorithm?

In a genetic algorithm, a population of strings (called chromosomes or the genotype of the genome), which encode candidate solutions (called individuals, creatures, or phenotypes) to an optimization problem, is evolved toward better solutions. Traditionally, solutions are represented in binary as strings of 0s and 1s, but other encodings are also possible. The evolution usually starts from a population of randomly generated individuals and happens in generations. In each generation, the fitness of every individual in the population is evaluated; multiple individuals are stochastically selected from the current population based on their fitness, and modified and possibly randomly mutated to form a new population. The new population is then used in the next iteration of the algorithm. Commonly, the algorithm terminates when either a maximum number of generations has been produced, or a satisfactory fitness level has been reached for the population. [5].

Basically a genetic algorithm requires:

1. a genetic representation of the solution domain,
2. a fitness function to evaluate the solution domain.

A standard representation of the solution is as an array of bits. Arrays of other types and structures can be used in essentially the same way. The main property that makes these genetic representations convenient is that their parts are easily aligned due to their fixed size, which facilitates simple crossover operations. Variable length representations may also be used, but crossover implementation is more complex in this case. Tree-like representations are explored in genetic programming and graph-form representations are explored in evolutionary programming; a mix of both linear chromosomes and trees is explored in gene expression programming.

The fitness function is defined over the genetic representation and measures the quality of the represented solution. The fitness function is always problem dependent.

Once the genetic representation and the fitness function are defined, a GA proceeds to initialize a population of solutions (usually randomly) and then (usually) to improve it through repetitive application of the mutation, crossover, inversion and selection operators. Uniform Crossover

In the crossover operation, two new children are formed by exchanging the genetic information between two parent chromosomes. Multipoint crossover defines crossover points as places between loci where an individual can be split. Uniform crossover generalizes this scheme to make every locus a potential crossover point. [4] A crossover mask, the same length as the individual structure is created at random and the parity of the bits in the mask indicate which parent will supply the offspring with which bits. This method is identical to discrete recombination. Consider the following two individuals with 11 binary variables each:

Individual 1  0 1 1 1 0 0 1 1 0 1 0
Individual 2  1 0 1 0 1 1 0 0 1 0 1

For each variable the parent who contributes its variable to the offspring is chosen randomly with equal probability. Here, the offspring 1 is produced by taking the bit from parent 1 if the corresponding mask bit is 1 or the bit from parent 2 if the corresponding mask bit is 0. Offspring 2 is created using the inverse of the mask, usually.

Sample 1  0 1 1 0 0 1 1 0 1 0
Sample 2  1 0 0 1 1 0 0 1 0 1

After crossover the new individuals are created:
offspring 1  1 1 1 0 1 1 1 1 1 1
offspring 2  0 0 1 1 0 0 0 0 0 0

Uniform 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 the individual bits in the
individual’s representation. How uniform crossover may be parameterized by applying a probability to the swapping of bits was demonstrated by William M. Spears et al.[2] 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. The chromosome cloning takes place when a pair of chromosomes does not crossover, thus creating offsprings that are exact copies of each parent.

The ultimate step in each generation is the mutation of individuals through the alteration of parts of their genes. Mutation alters a minute portion of a chromosome and thus institutes variability into the population of the subsequent generation. Mutation, a rarity in nature, denotes the alteration in the gene and assists us in avoiding loss of genetic diversity. Its chief intent is to ensure that the search algorithm is not bound on a local optimum. [1]

The Complete Process

Initialization
Initially many individual solutions are randomly generated to form an initial population. The population size depends on the nature of the problem, but typically contains several hundreds or thousands of possible solutions.

Selection
During each successive generation, a proportion of the existing population is selected to breed a new generation. Individual solutions are selected through a fitness-based process, where fitter solutions (as measured by a fitness function) are typically more likely to be selected. Certain selection methods rate the fitness of each solution and preferentially select the best solutions.

Reproduction
The next step is to generate a second generation population of solutions from those selected ones through genetic operators: crossover (also called recombination), and/or mutation.

For each new solution to be produced, a pair of "parent" solutions is selected for breeding from the pool selected previously. By producing a "child" solution using the above methods of crossover and mutation, a new solution is created which typically shares many of the characteristics of its "parents". New parents are selected for each new child, and the process continues until a new population of solutions of appropriate size is generated. These processes ultimately result in the next generation population of chromosomes that is different from the initial generation. Generally the average fitness will have increased by this procedure for the population, since only the best organisms from the first generation are selected.

This generational process is repeated until a termination condition has been reached. Common terminating conditions are:

- A solution is found that satisfies minimum criteria
- Fixed number of generations reached
- Allocated budget (computation time/money) reached
- The highest ranking solution's fitness is reaching or has reached a plateau such that successive iterations no longer produce better results
- Manual inspection

Genetic algorithm procedure (main steps)
Choose the initial population of individuals

Evaluate the fitness of each individual in that population

Repeat on this generation until termination (time limit, sufficient fitness achieved, etc.):
Applications of Genetic Algorithms
GAs have been used for problem-solving and for modelling. GAs are applied to many scientific, engineering problems, in business and entertainment, including:

Optimization: GAs have been used in a wide variety of optimisation tasks, including numerical optimisation, and combinatorial optimisation problems such as traveling salesman problem (TSP), circuit design, job shop scheduling and video & sound quality optimisation.

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 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-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, the evolution of communication, and trail-following behavior in ants.[7]

Criticisms
There are several criticisms of the use of a genetic algorithm compared to alternative optimization algorithms:

Repeated fitness function evaluation for complex problems are often the most prohibitive and limiting segment of artificial evolutionary algorithms. Finding the optimal solution to complex high dimensional, multimodal problems often requires very expensive fitness function evaluations. In real world problems such as structural optimization problems, one single function evaluation may require several hours to several days of complete simulation. Typical optimization methods can not deal with such types of problem

Genetic algorithms do not scale well with complexity. That is, where the number of elements which are exposed to mutation is large there is often an exponential increase in search space size. This makes it extremely difficult to use the technique on problems such as designing an engine, a house or plane. In order to make such problems tractable to evolutionary search, they must be broken down into the simplest representation possible.

The "better" solution is only in comparison to other solutions. As a result, the stop criterion is not clear in every problem.
In many problems, GAs may have a tendency to converge towards local optima or even arbitrary points rather than the global optimum of the problem. This means that it does not "know how" to sacrifice short-term fitness to gain longer-term fitness.

Several methods have been proposed to remedy this by increasing genetic diversity somehow and preventing early convergence, either by increasing the probability of mutation when the solution quality drops (called triggered hypermutation), or by occasionally introducing entirely new, randomly generated elements into the gene pool (called random immigrants). Again, evolution strategies and evolutionary programming can be implemented with a so-called "comma strategy" in which parents are not maintained and new parents are selected only from offspring. This can be more effective on dynamic problems.

Alternative and complementary algorithms include evolution strategies, evolutionary programming, simulated annealing, Gaussian adaptation, hill climbing, and swarm intelligence (e.g.: ant colony optimization, particle swarm optimization) and methods based on integer linear programming. [3]

**Conclusion**

Genetic Algorithms has been widely studied, experimented and applied in many fields in engineering worlds. Not only does GAs provide an alternative method to solving problem, it consistently outperforms other traditional methods in most of the problems link. Many of the real world problems involved finding optimal parameters, which might prove difficult for traditional methods but ideal for GAs. New algorithms are being explored and applied as a focused approach for problem solving nowadays.

**References:**