International Journal of Information Technology and Knowledge Management
July-December 2010, Volume 2, No. 2, pp. 451-454

GENETIC ALGORITHM: REVIEW AND APPLICATION

Manoj Kumar[1], Mohammad Husian[2], Naveen Upreti[3] & Deepti Gupta[4]

Genetic algorithms are considered as a search process used in computing to find exact or a approximate solution for optimization and search problems. There are also termed as global search heuristics. These techniques are inspired by evolutionary biology such as inheritance mutation, selection and cross over. These algorithms provide a technique for program to automatically improve their parameters. This paper is an introduction of genetic algorithm approach including various applications and described the integration of genetic algorithm with object oriented programming approaches.

Keywords: Genetic Algorithm, Chromosome, Evolutionary Algorithm, Selection, Mutation

1. INTRODUCTION

The Genetic algorithm is an adaptive heuristic search method based on population genetics. Genetic algorithm were introduced by John Holland in the early 1970s [1].Genetic algorithm is a probabilistic search algorithm based on the mechanics of natural selection and natural genetics. Genetic algorithm is started with a set of solutions called population. A solution is represented by a chromosome. The population size is preserved throughout each generation. At each generation, fitness of each chromosome is evaluated, and then chromosomes for the next generation are probabilistically selected according to their fitness values. Some of the selected chromosomes randomly mate and produce offspring. When producing offspring, crossover and mutation randomly occurs. Because chromosomes with high fitness values have high probability of being selected, chromosomes of the new generation may have higher average fitness value than those of the old generation. The process of evolution is repeated until the end condition is satisfied. The solutions in genetic algorithms are called chromosomes or strings [2]. In most cases, chromosomes are represented by lists or strings. Thus, many operations in genetic algorithm are operations on lists or strings. The very high level languages like Python or Perl are more productive in list processing or string processing than C/C++/Java. In bioinformatics, Python or Perl is widely used.

A genetic algorithm is a search technique used in computing to find exact or approximate solutions to optimization and search problems. Genetic algorithms are categorized as global search heuristics. Genetic algorithms are a particular class of evolutionary algorithms (EA) that use techniques inspired by evolutionary biology such as inheritance, mutation, selection, and crossover. Genetic algorithms have been used to find optimal solutions to complex problems in various domains such as biology, engineering, computer science, and social science. They have heuristic being used alternatively to simulated annealing, hill climbing, or tattoo search.

In contrast to local search methods, genetic algorithms are based on a set of independent computations controlled by a probabilistic strategy. This is a simulation of natural selection of best individuals inside successive generations. Following the classical terminology, a solution for a problem under consideration is called an individual. The set of considered individuals is called a population. Each individual has one chromosome string encoding its data characteristics. Then, a chromosome is a sequence of alleles representing one quantum of information, such as bit, digit, and letter etc. & is alternative data representation requires coding and decoding in order to exchange solutions with the nominal object space [3].

Genetic algorithms fall under the heading of evolutionary algorithm. Evolutionary algorithms are used to solve problems that do not already have a well defined efficient solution. Genetic algorithm have been used to solve optimization problems (scheduling, shortest path, etc), and in modeling systems where randomness is involved (e.g., the stock market).

Genetic programming is a way of evolving algorithms that will map data to a given result when no set formula is known. Mathematicians/programmers can easily find algorithms to solve problems that deal with 5 or so variables, but when the unknowns increase to 10, 20, 50 or more variables, the problem becomes close to impossible to solve.

[1,3,4]Department of Information Technology, International Institute for Special Education, Lucknow, INDIA

[2]Department of Information Technology,Azad Institute of Engineering & Technology, Lucknow, INDIA

Email: iisemanoj@gmail.com, mohd.husain90@gmail.com, naveenupreti@gmail.com, deepti_182@yahoo.com

In a case where the mathematical data is available and answers are available but the expression that joins the data to the answers is missing, a genetic algorithm can be used to 'evolve' an expression tree to create a very close fit to the data. Mutation, crossover, and all of the elements in genetic algorithms are used to breed the 'highest-fitness' tree for the given problem. At best, this will perfectly match the variables to the answer, other times it will generate an answer very close to the desired answer [4].

2. GENETIC ALGORITHM METHODOLOGY

2.1. 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. Traditionally, the population is generated randomly, covering the entire range of possible solutions (the search space). Occasionally, the solutions may be "seeded" in areas where optimal solutions are likely to be found.

2.2. 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. Other methods rate only a random sample of the population, as this process may be very time-consuming.

Most functions are stochastic and designed so that a small proportion of less fit solutions are selected. This helps keep the diversity of the population large, preventing premature convergence on poor solutions. Popular and well-studied selection methods include roulette wheel selection and tournament selection.

2.3. Reproduction

The next step is to generate a second generation population of solutions from those selected 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. Although reproduction methods that are based on the use of two parents are more "biology inspired", some research [5] suggests more than two "parents" are better to be used to reproduce a good quality chromosome.

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 or genetic algorithm from the first generation are selected for breeding, along with a small proportion of less fit solutions, for reasons already mentioned above.

2.4. Termination

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;
- Combinations of the above.

Simple Genetic algorithm pseudocodes:

1. Choose the initial population of individuals.
2. Evaluate the fitness of each individual in that population.
3. Repeat on this generation until termination: (time limit, sufficient fitness achieved, etc.)
   a. Select the best-fit individuals for reproduction.
   b. Breed new individuals through crossover and mutation operations to give birth to offspring.
   c. Evaluate the individual fitness of new individuals.
   d. Replace least-fit population with new individuals.

3. APPLICATION OF GENETIC ALGORITHM

3.1. Sensor Based Robot Path Planning

Yasuda and Takai applied genetic algorithm for sensor-based mobile robot path planning under unstructured environment in real-time [6]. After finding obstacles, the planning module generates a short and safe path to goal with obstacle avoidance, which is a sequence of control vectors of

orientation. With genetic algorithm, a path is represented as a set of orientation vectors with equal distance. Thus, the final path is the composition of polygonal lines (sum of vectors). To minimize the length, the change of orientation is restricted to 5 values from –45 deg to 45 deg. For fitness function, they used distance parameters between goals, obstacles. They used the combination of roulette and elite selection, one-point cross over. They tried to make their system simple to operate in real-time environment.

3.2. Image Processing

Gong and Yang applied genetic algorithm for stereo image processing [7]. Stereovision system generates disparity map; the disparity map should be smooth and detail. They used intensity-based approach. They increased the accuracy of the disparity map by removing the mismatches from occlusions and false targets. They formalized stereo matching as optimization problem; genetic algorithm optimized the compatibility between corresponding points and continuity of the disparity map. First, the 3D disparity is populated with dissimilarity values based on the source images; a fitness function is defined based on the Markov Random Field to test a disparity map. Genetic algorithm extracts best population from disparity map. Color image segmentation, graft crossover was applied; elitist strategy is applied for selection. Their experiment showed that genetic algorithm out performed existing methods.

3.3. Gaming

Genetic algorithms are excellent at searching very large problem spaces, and also for evolutionary development. For example, a large structure of possible traits for Quake II monsters (aggressiveness, probability of running away when low on health, probability of running away when few in numbers etc) can be created and then a genetic algorithm can be used to find the best combination of these structures to beat the player. A player would go through a level of the game and at the end, the program would pick the monsters that fared the best against the player, and use those in the next generation. Slowly, after a lot of playing, some reasonable characteristics would be evolved [8].

3.4. Real Time Systems

Sandstrom et al. applied genetic algorithm for assigning task priorities and offsets to guarantees that real time timing constraints [9]. Assigning timing constraint to task is not trivial problem in real-time system. They showed how timing constraints be mapped to attributes of periodic tasks running on standard pre-emptive RTOS (Real-Time Operating System) such as VxWorks and QNX. They used genetic algorithm because of the genetic algorithms ability to generate a result that satisfies a subset of the timing constraints in cases where it is impossible to fulfill all constraints. Genetic algorithm, the mechanism of natural selection, gradually improves individuals – timing constraints assignment - in a population. They have tested on a many test cases, and showed good result.

3.5. Job Shop Scheduling

Madureia et al. suggested genetic algorithm for the resolution of real world scheduling problems, and proposed a coordination mechanism [9]. Because of frequently changing dynamic environments, providing efficient production management and timely delivery are one of the hard to solve problems.

Scheduling is to allocate a set of machines to perform a set of jobs within a certain time period, and the goal of scheduling is to find an appropriate allocation – schedule – which maximizes certain performance measure. For the implementation issues, the solutions are encoded by natural representation, and the order crossover operator is used. They used the inversion mechanism as mutation operator.

Finally, Madureia et al. solved dynamic scheduling problem using a set of static scheduling by genetic algorithm, and they showed the feasibility of genetic algorithm in Job-Shop scheduling problem.

4. APPLYING OBJECT ORIENTED PROGRAMMING WITH GENETIC ALGORITHM

4.1. Evolvable Class Representation

Following previous work on the evolution of complete classes [10] we decided to represent an evolvable individual using a syntactic structure that couples a linear repository of class and instance variables representing the object state along with a set of evolvable methods (using an expression tree representation) that are responsible for the way an object acts and reacts, in terms of state changes and message passing.

4.2. State Representation

The use of memory in GP dates back to the work of Koza (1992), who used global registers that could be, manipulated with specially built storage operators. Teller (1994) introduced Indexed Memory to allow a selection from an arbitrary set of memory cells [11]. Additional read and write operations are made available in the language to allow this memory to be accessed and manipulated by various program parts. Koza (1999) went on and generalized both the above to the notion of an Automatically Defined Store [12]. Finally, Kirshenbaum (2000) presented work on the evolution of programs that use statically scoped local variables [13]. That is, variables whose visibility is bounded to a given scope – a subtree rooted on a Let construct. We employed a simple memory addressing scheme by combining type information

along with the mechanism of pass by reference [14] in order to operate on the object state space. The object memory is represented as a linked list of objects of interface type Settable, reminiscent of Teller's indexed memory but uses a different way to store and read values. References to these Settable objects are added to the language used to construct programs, and these represent the available instance variables (object state space).

Within a program structure these language elements are being passed by reference to specially built primitives that explicitly set the value of their argument. Once the method returns, the value of its argument will have been updated.

For our purposes a setValue (Settable s, double value) primitive method has been defined. Notice that the use of strong typing for drawing a distinction between settable variables and their underlying values allows for the emergence of various sorts of assignment schemes and obviates the need of devising a strategy to deal with illegal range of index values, a substantial issue when working with traditional indexed memory. We follow Teller's example and setValue is defined to return the original value held in the Settable object it has just overwritten.

4.3. Variation Operator

Our search employs two different variation schemes. These are Sub-tree macro mutation (MM - substituting a node in the tree with an entirely randomly generated sub-tree of the same return type, under depth or size constraints) and homologous Uniform Crossover (UXO) along with Point Mutation (PM) as defined by Poli and Langdon [15]. In the case of multi-tree programs the evolutionary algorithm must come to a decision as to which tree the variation operator will be applied. Each time the variation operator will be applied to the expression tree implementing the interface method drive with a probability of 1:0 and to each supplementary expression tree with a probability of 0:5. Additionally, for MM, other than choosing the tree node to be replaced at random we devised an additional simple node selection scheme that allows us to select nodes at different depth levels using a uniform probability distribution, with the expectation to render bigger changes more likely.

5. CONCLUSION & FUTURE SCOPE

Genetic algorithm is a probabilistic solving optimization problem which is modeled on a genetic evaluations process in biology and is focused as an effective algorithm to find a global optimum solution for many types of problem. This algorithm is extremely applicable in different artificial intelligence approaches as well as different basics approaches like object oriented, robotics and other in future we shall concentrate on the development of hybrid approaches using genetic algorithm an object oriented technology.

REFERENCE

[1] Tsoukalas, L., and Uhrig, R. Fuzzy and Neural Approaches in Engineering, Wiley, 1997.

[2] VWonjae Lee, Hak-Young Kim, "Genetic Algorithm Implementation in Python", Electronics and Telecommunications Research Institute.

[3] An Object-Oriented Environment for Specification and Concurrent Execution of Genetic Algorithms, Vancouver, British Columbia, Canada, 5-110 October 1992.

[4] "Introduction to Genetic algorithms", http://library.thinkquest.org/18242/ga.shtml

[5] Eiben, A. E. et al, "Genetic Algorithms with Multi-parent Recombination". PPSN, III: Proceedings of the International Conference on Evolutionary Computation. The Third Conference on Parallel Problem Solving from Nature: 78-87. ISBN 3-540-58484-6,1994

[6] Yasuda, G. and Takai, H. "Sensor-based Path Planning and Intelligent Steering Control of Nonholonomic Mobile Robots", Industrial Electronics Society, 2001. IECON '01. The 27th Annual Conference of the IEEE, 1, 2001 Page(s): 317 -322, 1, 2001.

[7] Minglun G. and Yee-Hong Y., "Multi-resolution Stereo Matching using Genetic Algorithm", Stereo and Multi-Baseline Vision, 2001. (SMBV 2001). Proceedings. IEEE Workshop on, pp 21 –29, 2001.

[8] Sandstrom, K.and Norstrom, C, "Managing Complex Temporal Requirements in Real-time Control Systems",. Engineering of Computer Based Systems, 2002. Proceedings, Ninth Annual IEEE International Conference and Workshop, Page(s): 103 –109, 2002.

[9] Madureira, A.; Ramos, C.; do Carmo Silva, S.,"A Coordination Mechanism for Real World Scheduling Problems using Genetic algorithms", Evolutionary Computation, 2002. CEC '02. Proceedings of the 2002 Congress on, 1, pp 175 –180, 2002.

[10] A. Agapitos and S. M. Lucas. "Evolving a Statistics Class using Object Oriented Evolutionary Programming". In Proceedings of the 10th European Conference on Genetic Programming, 2007.

[11] A. Teller. The Evolution of Mental Models. In K. E. Kinnear, Jr., editor, Advances in Genetic Programming, chapter 9, pages 199-219. MIT Press, 1994.

[12] J. R. Koza, D. Andre, F. H. Bennett III, and M. Keane. Genetic Programming 3: Darwinian Invention and Problem Solving. Morgan Kaufman.

[13] E. Kirshenbaum. "Genetic Programming with Statically Scoped Local Variables. Technical Report HPL- 2000-106, Hewlett Packard Laboratories, Palo Alto, 11 Aug. 2000.

[14] W. B. Langdon. Genetic Programming and Data Structures: Genetic Programming + Data Structures = Automatic Programming!, Volume 1 of Genetic Programming. Kluwer, Boston, 24 Apr. 1998.

[15] R. Poli and W. B. Langdon. Genetic Programming with One-point Crossover and Point Mutation. Technical Report CSRP-97-13, University of Birmingham, School of Computer Science, Birmingham, B15 2TT, UK, 15 Apr. 1997.