Elements of Genetic Algorithms

Genetic algorithm

In computer science and operations research, a **genetic algorithm** (**GA**) is a metaheuristic inspired by the process of natural selection that belongs to the larger class of evolutionary algorithms (EA). Genetic algorithms are commonly used to generate high-quality solutions to optimization and search problems by relying on biologically inspired operators such as mutation, crossover and selection.

Methodology

Optimization problems

In a genetic algorithm, a population of candidate solutions (called individuals, creatures, or phenotypes) to an optimization problem is evolved toward better solutions. Each candidate solution has a set of properties (its chromosomes or genotype) which can be mutated and altered; 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 is an iterative process, with the population in each iteration called a *generation*. In each generation, the fitness of every individual in the population is evaluated; the fitness is usually the value of the objective function in the optimization problem being solved. The more fit individuals are stochastically selected from the current population, and each individual's genome is modified (recombined and possibly randomly mutated) to form a new generation. The new generation of candidate solutions 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.

A typical genetic algorithm requires:

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

A standard representation of each candidate 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. Once the genetic representation and the fitness function are defined, a GA proceeds to initialize a population of solutions and then to improve it through repetitive application of the mutation, crossover, inversion and selection operators.

Initialization

The population size depends on the nature of the problem, but typically contains several hundreds or thousands of possible solutions. Often, the initial population is generated randomly, allowing 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.

Selection

During each successive generation, a portion 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 the former process may be very time-consuming. The fitness function is defined over the genetic representation and measures the *quality* of the represented solution. The fitness function is always problem dependent. For instance, in the knapsack problem one wants to maximize the total value of objects that can be put in a knapsack of some fixed capacity. A representation of a solution might be an array of bits, where each bit represents a different object, and the value of the bit (0 or 1) represents whether or not the object is in the knapsack. Not every such representation is valid, as the size of objects may exceed the capacity of the knapsack. The *fitness* of the solution is the sum of values of all objects in the knapsack if the representation is valid, or 0 otherwise. In some problems, it is hard or even impossible to define the fitness expression; in these cases, a simulation may be used to determine the fitness function value of a phenotype (e.g. computational fluid dynamics is used to determine the air resistance of a vehicle whose shape is encoded as the phenotype), or even interactive genetic algorithms are used.

Genetic operators

The next step is to generate a second generation population of solutions from those selected through a combination of genetic operators: crossover (also called recombination), and 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 suggests that more than two "parents" generate higher quality chromosomes. 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 for breeding, along with a small proportion of less fit solutions. These less fit solutions ensure genetic diversity within the genetic pool of the parents and therefore ensure the genetic diversity of the subsequent generation of children.

Heuristics

In addition to the main operators above, other heuristics may be employed to make the calculation faster or more robust. The *speciation* heuristic penalizes crossover between candidate solutions that are too similar; this encourages population diversity and helps prevent premature convergence to a less optimal solution.

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

The building block hypothesis

Genetic algorithms are simple to implement, but their behavior is difficult to understand. In particular, it is difficult to understand why these algorithms frequently succeed at generating solutions of high fitness when applied to practical problems. The building block hypothesis (BBH) consists of:

1. A description of a heuristic that performs adaptation by identifying and recombining "building blocks", i.e. low order, low defining-length schemata with above average fitness.
2. A hypothesis that a genetic algorithm performs adaptation by implicitly and efficiently implementing this heuristic.

Despite the lack of consensus regarding the validity of the building-block hypothesis, it has been consistently evaluated and used as reference throughout the years. Many estimation of distribution algorithms, for example, have been proposed in an attempt to provide an environment in which the hypothesis would hold. Although good results have been reported for some classes of problems, skepticism concerning the generality and/or practicality of the building-block hypothesis as an explanation for GAs efficiency still remains. Indeed, there is a reasonable amount of work that attempts to understand its limitations from the perspective of estimation of distribution algorithms.

Limitations

There are limitations of the use of a genetic algorithm compared to alternative optimization algorithms:

- Repeated fitness function evaluation for complex problems is 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, a single function evaluation may require several hours to several days of complete simulation. Typical optimization methods cannot deal with such types of problem. In this case, it may be necessary to forgo an exact evaluation and use an approximated fitness that is computationally efficient. It is apparent that amalgamation of approximate models may be one of the most promising approaches to convincingly use GA to solve complex real life problems.
- 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. In order to make such problems tractable to evolutionary search, they must be broken down into the simplest representation possible. Hence, we typically see evolutionary algorithms encoding designs for fan blades instead of engines, building shapes instead of detailed construction plans, and air foils instead of whole aircraft designs. The second problem of complexity is the issue of how to protect parts that have evolved to represent good solutions from further destructive mutation, particularly when their fitness assessment requires them to combine well with other parts.
- 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 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. The likelihood of this occurring depends on the shape of the fitness landscape: certain problems may provide an easy ascent towards a global optimum; others may make it easier for the function to find the local optima. This problem may be alleviated by using a different fitness function, increasing the rate of mutation, or by using selection techniques that maintain a diverse population of solutions, although the No Free Lunch theorem proves that there is no general solution to this problem. A common technique to maintain diversity is to impose a "niche penalty", wherein, any group of individuals of sufficient similarity (niche radius) have a penalty added, which will reduce the representation of that group in subsequent generations, permitting other (less similar) individuals to be maintained in the population. This trick, however, may not be effective, depending on the landscape of the problem. Another possible technique would be to simply replace part of the population with randomly generated individuals, when most of the population is too similar to each other. Diversity is important in genetic algorithms (and genetic programming) because crossing over a homogeneous population does not yield new solutions. In evolution strategies and

evolutionary programming, diversity is not essential because of a greater reliance on mutation.

- Operating on dynamic data sets is difficult, as genomes begin to converge early on towards solutions which may no longer be valid for later data. 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.
- GAs cannot effectively solve problems in which the only fitness measure is a single right/wrong measure (like decision problems), as there is no way to converge on the solution (no hill to climb). In these cases, a random search may find a solution as quickly as a GA. However, if the situation allows the success/failure trial to be repeated giving (possibly) different results, then the ratio of successes to failures provides a suitable fitness measure.
- For specific optimization problems and problem instances, other optimization algorithms may be more efficient than genetic algorithms in terms of speed of convergence. 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. The suitability of genetic algorithms is dependent on the amount of knowledge of the problem; well known problems often have better, more specialized approaches.

Variants

Chromosome representation

The simplest algorithm represents each chromosome as a bit string. Typically, numeric parameters can be represented by integers, though it is possible to use floating point representations. The floating-point representation is natural to evolution strategies and evolutionary programming. The notion of real-valued genetic algorithms has been offered but is really a misnomer because it does not really represent the building block theory that was proposed by John Henry Holland in the 1970s. This theory is not without support though, based on theoretical and experimental results (see below). The basic algorithm performs crossover and mutation at the bit level. Other variants treat the chromosome as a list of numbers which are indexes into an instruction table, nodes in a linked list, hashes, objects, or any other imaginable data structure. Crossover and mutation are performed so as to respect data element boundaries. For most data types, specific variation operators can be designed. Different chromosomal data types seem to work better or worse for different specific problem domains. When bit-string representations of integers are used, Gray coding is often employed. In this way, small changes in the integer can be readily affected through mutations

or crossovers. This has been found to help prevent premature convergence at so-called *Hamming walls*, in which too many simultaneous mutations (or crossover events) must occur in order to change the chromosome to a better solution. Other approaches involve using arrays of real-valued numbers instead of bit strings to represent chromosomes. Results from the theory of schemata suggest that in general the smaller the alphabet, the better the performance, but it was initially surprising to researchers that good results were obtained from using real-valued chromosomes. This was explained as the set of real values in a finite population of chromosomes as forming a *virtual alphabet* (when selection and recombination are dominant) with a much lower cardinality than would be expected from a floating-point representation. An expansion of the Genetic Algorithm accessible problem domain can be obtained through more complex encoding of the solution pools by concatenating several types of heterogenously encoded genes into one chromosome. This particular approach allows for solving optimization problems that require vastly disparate definition domains for the problem parameters. For instance, in problems of cascaded controller tuning, the internal loop controller structure can belong to a conventional regulator of three parameters, whereas the external loop could implement a linguistic controller (such as a fuzzy system) which has an inherently different description. This particular form of encoding requires a specialized crossover mechanism that recombines the chromosome by section, and it is a useful tool for the modelling and simulation of complex adaptive systems, especially evolution processes.

Elitism

A practical variant of the general process of constructing a new population is to allow the best organism(s) from the current generation to carry over to the next, unaltered. This strategy is known as *elitist selection* and guarantees that the solution quality obtained by the GA will not decrease from one generation to the next.

Parallel implementations

Parallel implementations of genetic algorithms come in two flavours. Coarse-grained parallel genetic algorithms assume a population on each of the computer nodes and migration of individuals among the nodes. Fine-grained parallel genetic algorithms assume an individual on each processor node which acts with neighbouring individuals for selection and reproduction. Other variants, like genetic algorithms for online optimization problems, introduce time-dependence or noise in the fitness function.

Adaptive GAs

Genetic algorithms with adaptive parameters (adaptive genetic algorithms, AGAs) is another significant and promising variant of genetic algorithms. The probabilities of crossover (pc) and mutation (pm) greatly determine the degree of solution accuracy and the convergence speed that genetic algorithms can obtain. Instead of using fixed values of *pc* and *pm*, AGAs utilize the population information in each generation and adaptively adjust the *pc* and *pm* in order to maintain the population diversity as well as to sustain the convergence capacity. In

AGA (adaptive genetic algorithm), the adjustment of *pc* and *pm* depends on the fitness values of the solutions. In *CAGA* (clustering-based adaptive genetic algorithm), through the use of clustering analysis to judge the optimization states of the population, the adjustment of *pc* and *pm* depends on these optimization states. It can be quite effective to combine GA with other optimization methods. GA tends to be quite good at finding generally good global solutions, but quite inefficient at finding the last few mutations to find the absolute optimum. Other techniques (such as simple hill climbing) are quite efficient at finding absolute optimum in a limited region.

This means that the rules of genetic variation may have a different meaning in the natural case. For instance – provided that steps are stored in consecutive order – crossing over may sum a number of steps from maternal DNA adding a number of steps from paternal DNA and so on. This is like adding vectors that more probably may follow a ridge in the phenotypic landscape. Thus, the efficiency of the process may be increased by many orders of magnitude. Moreover, the inversion operator has the opportunity to place steps in consecutive order or any other suitable order in favour of survival or efficiency. A variation, where the population as a whole is evolved rather than its individual members, is known as gene pool recombination. A number of variations have been developed to attempt to improve performance of GAs on problems with a high degree of fitness epistasis, i.e. where the fitness of a solution consists of interacting subsets of its variables.

References

Eiben, A. E. et al (1994). "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.

Ting, Chuan-Kang (2005). "On the Mean Convergence Time of Multi-parent Genetic Algorithms Without Selection". Advances in Artificial Life: 403–412. ISBN 978-3-540-28848-0.

Shir, Ofer M. (2012). "Niching in Evolutionary Algorithms". In Rozenberg, Grzegorz; Bäck, Thomas; Kok, Joost N. (eds.). Handbook of Natural Computing. Springer Berlin Heidelberg. pp. 1035–1069. doi:10.1007/978-3-540-92910-9_32. ISBN 9783540929093.

Harik, Georges R.; Lobo, Fernando G.; Sastry, Kumara (1 January 2006). Linkage Learning via Probabilistic Modeling in the Extended Compact Genetic Algorithm (ECGA). Scalable Optimization Via Probabilistic Modeling. Studies in Computational Intelligence. 33. pp. 39–61

Pelikan, Martin; Goldberg, David E.; Cantú-Paz, Erick (1 January 1999). BOA: The Bayesian Optimization Algorithm. Proceedings of the 1st Annual Conference on Genetic and Evolutionary Computation - Volume 1. Gecco'99. pp. 525–532. ISBN 9781558606111.

Sadowski, Krzysztof L.; Bosman, Peter A.N.; Thierens, Dirk (1 January 2013). On the Usefulness of Linkage Processing for Solving MAX-SAT. Proceedings of the 15th Annual Conference on Genetic and Evolutionary Computation. Gecco '13. pp. 853–860.

Goldberg, David E. (1991). "The theory of virtual alphabets". Parallel Problem Solving from Nature. Parallel Problem Solving from Nature, Lecture Notes in Computer Science. Lecture Notes in Computer Science.

Zhang, J.; Chung, H.; Lo, W. L. (2007). "Clustering-Based Adaptive Crossover and Mutation Probabilities for Genetic Algorithms". IEEE Transactions on Evolutionary Computation