Genetic algorithm

From Wikipedia, the free encyclopedia

In the computer science field of artificial intelligence, a **genetic algorithm (GA)** is a search heuristic that mimics the process of natural selection. This heuristic (also sometimes called a metaheuristic) is routinely used to generate useful solutions to optimization and search problems.[1] Genetic algorithms belong to the larger class of evolutionary algorithms (EA), which generate solutions to optimization problems using techniques inspired by natural evolution, such as inheritance, mutation, selection, and crossover.

Genetic algorithms find application in bioinformatics, phylogenetics, computational science, engineering, economics, chemistry, manufacturing, mathematics, physics, pharmacometrics and other fields.

The 2006 NASA ST5 spacecraft antenna. This complicated shape was found by an evolutionary computer design program to create the best radiation pattern.

Contents

Methodology

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.[2]

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.[2] 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 of genetic algorithm

Initially many individual solutions are (usually) 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, 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 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 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[3][4] 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.

Opinion is divided over the importance of crossover versus mutation. There are many references in Fogel (2006) that support the importance of mutation-based search.

Although crossover and mutation are known as the main genetic operators, it is possible to use other operators such as regrouping, colonization-extinction, or migration in genetic algorithms.[5]

It is worth tuning parameters such as the mutation probability, crossover probability and population size to find reasonable settings for the problem class being worked on. A very small mutation rate may lead to genetic drift (which is non-ergodic in nature). A recombination rate that is too high may lead to premature convergence of the genetic algorithm. A mutation rate that is too high may lead to loss of good solutions, unless elitist selection is employed.

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.

Goldberg describes the heuristic as follows:

> "Short, low order, and highly fit schemata are sampled, recombined [crossed over], and resampled to form strings of potentially higher fitness. In a way, by working with these particular schemata [the building blocks], we have reduced the complexity of our problem; instead of building high-performance strings by trying every conceivable combination, we construct better and better strings from the best partial solutions of past samplings.
>
> "Because highly fit schemata of low defining length and low order play such an important role in the action of genetic algorithms, we have already given them a special name: building blocks. Just as a child creates magnificent fortresses through the arrangement of simple blocks of wood, so does a genetic algorithm seek near optimal performance through the juxtaposition of short, low-order, high-performance schemata, or building blocks."[6]

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 can not 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. 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. Hence we typically see evolutionary algorithms encoding designs for fan blades instead of engines, building shapes instead of detailed construction plans, airfoils 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 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. 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,[7] although the No Free Lunch theorem[8] 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 effected 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.[9][10]

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.[11]

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 neighboring 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),[12] the adjustment of pc and pm depends on the fitness values of the solutions. In *CAGA* (clustering-based adaptive genetic algorithm),[13] 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. Alternating GA and hill climbing can improve the efficiency of GA while overcoming the lack of robustness of hill climbing.

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. (See for instance [14] or example in travelling salesman problem, in particular the use of an edge recombination operator.)

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. Such algorithms aim to learn (before exploiting) these beneficial phenotypic interactions. As such, they are aligned with the Building Block Hypothesis in adaptively reducing disruptive recombination. Prominent examples of this approach include the mGA,[15] GEMGA[16] and LLGA.[17]

Problem domains

Problems which appear to be particularly appropriate for solution by genetic algorithms include timetabling

and scheduling problems, and many scheduling software packages are based on GAs. GAs have also been applied to engineering.[18] Genetic algorithms are often applied as an approach to solve global optimization problems.

As a general rule of thumb genetic algorithms might be useful in problem domains that have a complex fitness landscape as mixing, i.e., mutation in combination with crossover, is designed to move the population away from local optima that a traditional hill climbing algorithm might get stuck in. Observe that commonly used crossover operators cannot change any uniform population. Mutation alone can provide ergodicity of the overall genetic algorithm process (seen as a Markov chain).

Examples of problems solved by genetic algorithms include: mirrors designed to funnel sunlight to a solar collector,[19] antennae designed to pick up radio signals in space,[20] and walking methods for computer figures.[21]

In his *Algorithm Design Manual*, Skiena advises against genetic algorithms for any task:

> [I]t is quite unnatural to model applications in terms of genetic operators like mutation and crossover on bit strings. The pseudobiology adds another level of complexity between you and your problem. Second, genetic algorithms take a very long time on nontrivial problems. [...] [T]he analogy with evolution—where significant progress require millions of years—can be quite appropriate.
>
> [...]
>
> I have never encountered any problem where genetic algorithms seemed to me the right way to attack it. Further, I have never seen any computational results reported using genetic algorithms that have favorably impressed me. Stick to simulated annealing for your heuristic search voodoo needs.
>
> —Steven Skiena[22]:267

History

In 1950, Alan Turing proposed a "learning machine" which would parallel the principles of evolution.[23] Computer simulation of evolution started as early as in 1954 with the work of Nils Aall Barricelli, who was using the computer at the Institute for Advanced Study in Princeton, New Jersey.[24][25] His 1954 publication was not widely noticed. Starting in 1957,[26] 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)[27] and Crosby (1973).[28] 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.[29] Other noteworthy early pioneers include Richard Friedberg, George Friedman, and Michael Conrad. Many early papers are reprinted by Fogel (1998).[30]

Although Barricelli, in work he reported in 1963, had simulated the evolution of ability to play a simple game,[31] 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.[32][33][34][35] 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). His work originated with studies of cellular automata, conducted by Holland and his students at the University of Michigan. Holland introduced a formalized framework for predicting the quality of the next generation, known as Holland's Schema Theorem. Research in GAs remained largely theoretical until the mid-1980s, when The First International Conference on Genetic Algorithms was held in Pittsburgh, Pennsylvania.

As academic interest grew, the dramatic increase in desktop computational power allowed for practical application of the new technique. In the late 1980s, General Electric started selling the world's first genetic algorithm product, a mainframe-based toolkit designed for industrial processes. In 1989, Axcelis, Inc. released Evolver, the world's first commercial GA product for desktop computers. The New York Times technology writer John Markoff wrote[36] about Evolver in 1990, and it remained the only interactive commercial genetic algorithm until 1995.[37] Evolver was sold to Palisade in 1997, translated into several languages, and is currently in its 6th version.[38]

Related techniques

Parent fields

Genetic algorithms are a sub-field of:

- Evolutionary algorithms
- Evolutionary computing
- Metaheuristics
- Stochastic optimization
- Optimization

Related fields

Evolutionary algorithms

Evolutionary algorithms is a sub-field of evolutionary computing.

- Evolution strategies (ES, see Rechenberg, 1994) evolve individuals by means of mutation and intermediate or discrete recombination. ES algorithms are designed particularly to solve problems in the real-value domain. They use self-adaptation to adjust control parameters of the search. De-randomization of self-adaptation has led to the contemporary Covariance Matrix Adaptation Evolution Strategy (CMA-ES).

- Evolutionary programming (EP) involves populations of solutions with primarily mutation and selection and arbitrary representations. They use self-adaptation to adjust parameters, and can include other variation operations such as combining information from multiple parents.

- Gene expression programming (GEP) also uses populations of computer programs. These complex computer programs are encoded in simpler linear chromosomes of fixed length, which are afterwards expressed as expression trees. Expression trees or computer programs evolve because the chromosomes undergo mutation and recombination in a manner similar to the canonical GA. But thanks to the special organization of GEP chromosomes, these genetic modifications always result in valid computer programs.[39]

- Genetic programming (GP) is a related technique popularized by John Koza in which computer programs, rather than function parameters, are optimized. Genetic programming often uses tree-based internal data structures to represent the computer programs for adaptation instead of the list structures typical of genetic algorithms.

- Grouping genetic algorithm (GGA) is an evolution of the GA where the focus is shifted from individual items, like in classical GAs, to groups or subset of items.[40] The idea behind this GA evolution proposed by Emanuel Falkenauer is that solving some complex problems, a.k.a. *clustering* or *partitioning* problems where a set of items must be split into disjoint group of items in an optimal way, would better be achieved by making characteristics of the groups of items equivalent to genes. These kind of problems include bin packing, line balancing, clustering with respect to a distance measure, equal piles, etc., on which classic GAs proved to perform poorly. Making genes equivalent to groups implies chromosomes that are in general of variable length, and special genetic operators that manipulate whole groups of items. For bin packing in particular, a GGA hybridized with the Dominance Criterion of Martello and Toth, is arguably the best technique to date.

- Interactive evolutionary algorithms are evolutionary algorithms that use human evaluation. They are usually applied to domains where it is hard to design a computational fitness function, for example, evolving images, music, artistic designs and forms to fit users' aesthetic preference.

Swarm intelligence

Swarm intelligence is a sub-field of evolutionary computing.

- Ant colony optimization (ACO) uses many ants (or agents) to traverse the solution space and find locally productive areas. While usually inferior to genetic algorithms and other forms of local search, it is able to produce results in problems where no global or up-to-date perspective can be obtained, and thus the other methods cannot be applied.

- Particle swarm optimization (PSO) is a computational method for multi-parameter optimization which also uses population-based approach. A population (swarm) of candidate solutions (particles) moves in the search space, and the movement of the particles is influenced both by their own best known position and swarm's global best known position. Like genetic algorithms, the PSO method depends on information sharing among population members. In some problems the PSO is often more computationally efficient than the GAs, especially in unconstrained problems with continuous

variables.[41]

- Intelligent Water Drops or the IWD algorithm [42] is a nature-inspired optimization algorithm inspired from natural water drops which change their environment to find the near optimal or optimal path to their destination. The memory is the river's bed and what is modified by the water drops is the amount of soil on the river's bed.

Other evolutionary computing algorithms

Evolutionary computation is a sub-field of the metaheuristic methods.

- Harmony search (HS) is an algorithm mimicking the behaviour of musicians in the process of improvisation.
- Memetic algorithm (MA), often called *hybrid genetic algorithm* among others, is a population-based method in which solutions are also subject to local improvement phases. The idea of memetic algorithms comes from memes, which unlike genes, can adapt themselves. In some problem areas they are shown to be more efficient than traditional evolutionary algorithms.
- Bacteriologic algorithms (BA) inspired by evolutionary ecology and, more particularly, bacteriologic adaptation. Evolutionary ecology is the study of living organisms in the context of their environment, with the aim of discovering how they adapt. Its basic concept is that in a heterogeneous environment, you can't find one individual that fits the whole environment. So, you need to reason at the population level. It is also believed BAs could be successfully applied to complex positioning problems (antennas for cell phones, urban planning, and so on) or data mining.[43]
- Cultural algorithm (CA) consists of the population component almost identical to that of the genetic algorithm and, in addition, a knowledge component called the belief space.
- Differential Search Algorithm (DS) inspired by migration of superorganisms.[44]
- Gaussian adaptation (normal or natural adaptation, abbreviated NA to avoid confusion with GA) is intended for the maximisation of manufacturing yield of signal processing systems. It may also be used for ordinary parametric optimisation. It relies on a certain theorem valid for all regions of acceptability and all Gaussian distributions. The efficiency of NA relies on information theory and a certain theorem of efficiency. Its efficiency is defined as information divided by the work needed to get the information.[45] Because NA maximises mean fitness rather than the fitness of the individual, the landscape is smoothed such that valleys between peaks may disappear. Therefore it has a certain “ambition” to avoid local peaks in the fitness landscape. NA is also good at climbing sharp crests by adaptation of the moment matrix, because NA may maximise the disorder (average information) of the Gaussian simultaneously keeping the mean fitness constant.

Other metaheuristic methods

Metaheuristic methods broadly fall within stochastic optimisation methods.

- Simulated annealing (SA) is a related global optimization technique that traverses the search space by testing random mutations on an individual solution. A mutation that increases fitness is always accepted. A mutation that lowers fitness is accepted probabilistically based on the difference in fitness and a decreasing temperature parameter. In SA parlance, one speaks of seeking the lowest energy instead of the maximum fitness. SA can also be used within a standard GA algorithm by starting with a relatively high rate of mutation and decreasing it over time along a given schedule.

- Tabu search (TS) is similar to simulated annealing in that both traverse the solution space by testing mutations of an individual solution. While simulated annealing generates only one mutated solution, tabu search generates many mutated solutions and moves to the solution with the lowest energy of those generated. In order to prevent cycling and encourage greater movement through the solution space, a tabu list is maintained of partial or complete solutions. It is forbidden to move to a solution that contains elements of the tabu list, which is updated as the solution traverses the solution space.

- Extremal optimization (EO) Unlike GAs, which work with a population of candidate solutions, EO evolves a single solution and makes local modifications to the worst components. This requires that a suitable representation be selected which permits individual solution components to be assigned a quality measure ("fitness"). The governing principle behind this algorithm is that of *emergent* improvement through selectively removing low-quality components and replacing them with a randomly selected component. This is decidedly at odds with a GA that selects good solutions in an attempt to make better solutions.

Other stochastic optimisation methods

- The cross-entropy (CE) method generates candidates solutions via a parameterized probability distribution. The parameters are updated via cross-entropy minimization, so as to generate better samples in the next iteration.

- Reactive search optimization (RSO) advocates the integration of sub-symbolic machine learning techniques into search heuristics for solving complex optimization problems. The word reactive hints at a ready response to events during the search through an internal online feedback loop for the self-tuning of critical parameters. Methodologies of interest for Reactive Search include machine learning and statistics, in particular reinforcement learning, active or query learning, neural networks, and meta-heuristics.

See also

- List of genetic algorithm applications
- Propagation of schema
- Universal Darwinism

- Metaheuristics

References

1. ^ Mitchell 1996, p. 2.
2. ^ ***a b*** Whitley 1994, p. 66.
3. ^ 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.
4. ^ 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.
5. ^ Akbari, Ziarati (2010). "A multilevel evolutionary algorithm for optimizing numerical functions" IJIEC 2 (2011): 419–430 [1] (http://growingscience.com/ijiec/Vol2/IJIEC_2010_11.pdf)
6. ^ Goldberg 1989, p. 41.
7. ^ Taherdangkoo, Mohammad; Paziresh, Mahsa; Yazdi, Mehran; Bagheri, Mohammad Hadi (19 November 2012). "An efficient algorithm for function optimization: modified stem cells algorithm". *Central European Journal of Engineering* **3** (1): 36–50. doi:10.2478/s13531-012-0047-8 (http://dx.doi.org/10.2478%2Fs13531-012-0047-8).
8. ^ Wolpert, D.H., Macready, W.G., 1995. No Free Lunch Theorems for Optimisation. Santa Fe Institute, SFI-TR-05-010, Santa Fe.
9. ^ Goldberg, David E. (1991). "The theory of virtual alphabets" (http://link.springer.com/chapter/10.1007/BFb0029726). *Parallel Problem Solving from Nature, Lecture Notes in Computer Science* **496**: 13–22. doi:10.1007/BFb0029726 (http://dx.doi.org/10.1007%2FBFb0029726). Retrieved 2 July 2013.
10. ^ Janikow, C. Z.; Michalewicz, Z. (1991). "An Experimental Comparison of Binary and Floating Point Representations in Genetic Algorithms" (http://www.cs.umsl.edu/~janikow/publications/1991/GAbin/text.pdf). *Proceedings of the Fourth International Conference on Genetic Algorithms*: 31–36. Retrieved 2 July 2013.
11. ^ Baluja, Shumeet; Caruana, Rich (1995). "Removing the genetics from the standard genetic algorithm" (http://www.ri.cmu.edu/pub_files/pub2/baluja_shumeet_1995_1/baluja_shumeet_1995_1.pdf). ICML.
12. ^ Srinivas. M and Patnaik. L, "Adaptive probabilities of crossover and mutation in genetic algorithms," IEEE Transactions on System, Man and Cybernetics, vol.24, no.4, pp.656–667, 1994. (http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=286385)
13. ^ ZHANG. J, Chung. H and Lo. W. L, “Clustering-Based Adaptive Crossover and Mutation Probabilities for Genetic Algorithms”, IEEE Transactions on Evolutionary Computation vol.11, no.3, pp. 326–335, 2007. (http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=4220690)
14. ^ Evolution-in-a-nutshell (http://web.telia.com/~u91131915/traveller.htm)
15. ^ D.E. Goldberg, B. Korb, and K. Deb. "Messy genetic algorithms: Motivation, analysis, and first results". Complex Systems, 5(3):493–530, October 1989. (http://www.complex-systems.com/issues/03-5.html)
16. ^ Gene expression: The missing link in evolutionary computation
17. ^ G. Harik. Learning linkage to efficiently solve problems of bounded difficulty using genetic algorithms. PhD thesis, Dept. Computer Science, University of Michigan, Ann Arbour, 1997 (http://portal.acm.org/citation.cfm?id=269517)
18. ^ Tomoiagă B, Chindriş M, Sumper A, Sudria-Andreu A, Villafafila-Robles R. Pareto Optimal Reconfiguration of Power Distribution Systems Using a Genetic Algorithm Based on NSGA-II. (http://www.mdpi.com/1996-1073/6/3/1439/pdf) Energies. 2013; 6(3):1439-1455.

19. ^ Gross, Bill. "A solar energy system that tracks the sun" (http://www.ted.com/talks/bill_gross_on_new_energy.html). *TED*. Retrieved 20 November 2013.
20. ^ Hornby, G. S.; Linden, D. S.; Lohn, J. D., *Automated Antenna Design with Evolutionary Algorithms* (http://ti.arc.nasa.gov/m/pub-archive/1244h/1244%20(Hornby).pdf)
21. ^ http://goatstream.com/research/papers/SA2013/index.html
22. ^ Skiena, Steven (2010). *The Algorithm Design Manual* (2nd ed.). Springer Science+Business Media. ISBN 1-849-96720-2.
23. ^ Turing, Alan M. "Computing machinery and intelligence" (http://mind.oxfordjournals.org/content/LIX/236/433). *Mind* **LIX** (238): 433–460. doi:10.1093/mind/LIX.236.433 (http://dx.doi.org/10.1093%2Fmind%2FLIX.236.433).
24. ^ Barricelli, Nils Aall (1954). "Esempi numerici di processi di evoluzione". *Methodos*: 45–68.
25. ^ Barricelli, Nils Aall (1957). "Symbiogenetic evolution processes realized by artificial methods". *Methodos*: 143–182.
26. ^ Fraser, Alex (1957). "Simulation of genetic systems by automatic digital computers. I. Introduction". *Aust. J. Biol. Sci.* **10**: 484–491.
27. ^ Fraser, Alex; Burnell, Donald (1970). *Computer Models in Genetics*. New York: McGraw-Hill. ISBN 0-07-021904-4.
28. ^ Crosby, Jack L. (1973). *Computer Simulation in Genetics*. London: John Wiley & Sons. ISBN 0-471-18880-8.
29. ^ 02.27.96 - UC Berkeley's Hans Bremermann, professor emeritus and pioneer in mathematical biology, has died at 69 (http://berkeley.edu/news/media/releases/96legacy/releases.96/14319.html)
30. ^ Fogel, David B. (editor) (1998). *Evolutionary Computation: The Fossil Record*. New York: IEEE Press. ISBN 0-7803-3481-7.
31. ^ Barricelli, Nils Aall (1963). "Numerical testing of evolution theories. Part II. Preliminary tests of performance, symbiogenesis and terrestrial life". *Acta Biotheoretica* (16): 99–126.
32. ^ Rechenberg, Ingo (1973). *Evolutionsstrategie*. Stuttgart: Holzmann-Froboog. ISBN 3-7728-0373-3.
33. ^ Schwefel, Hans-Paul (1974). *Numerische Optimierung von Computer-Modellen (PhD thesis)*.
34. ^ Schwefel, Hans-Paul (1977). *Numerische Optimierung von Computor-Modellen mittels der Evolutionsstrategie : mit einer vergleichenden Einführung in die Hill-Climbing- und Zufallsstrategie*. Basel; Stuttgart: Birkhäuser. ISBN 3-7643-0876-1.
35. ^ Schwefel, Hans-Paul (1981). *Numerical optimization of computer models (Translation of 1977 Numerische Optimierung von Computor-Modellen mittels der Evolutionsstrategie*. Chichester ; New York: Wiley. ISBN 0-471-09988-0.
36. ^ Markoff, John (29 August 1990). "What's the Best Answer? It's Survival of the Fittest" (http://www.nytimes.com/1990/08/29/business/business-technology-what-s-the-best-answer-it-s-survival-of-the-fittest.html). New York Times. Retrieved 9 August 2009.
37. ^ Ruggiero, Murray A.. (2009-08-01) Fifteen years and counting (http://www.futuresmag.com/2009/08/01/fifteen-years-and-counting?t=technology&page=2). Futuresmag.com. Retrieved on 2013-08-07.
38. ^ Evolver: Sophisticated Optimization for Spreadsheets (http://www.palisade.com/evolver/). Palisade. Retrieved on 2013-08-07.
39. ^ Ferreira, C. "Gene Expression Programming: A New Adaptive Algorithm for Solving Problems" (http://www.gene-expression-programming.com/webpapers/GEP.pdf). Complex Systems, Vol. 13, issue 2: 87-129.
40. ^ Falkenauer, Emanuel (1997). *Genetic Algorithms and Grouping Problems*. Chichester, England: John Wiley & Sons Ltd. ISBN 978-0-471-97150-4.

41. ^ Rania Hassan, Babak Cohanim, Olivier de Weck, Gerhard Vente r (2005) A comparison of particle swarm optimization and the genetic algorithm (http://www.mit.edu/~deweck/PDF_archive /3%20Refereed%20Conference/3_50_AIAA-2005-1897.pdf)
42. ^ Hamed Shah-Hosseini, The intelligent water drops algorithm: a nature-inspired swarm-based optimization algorithm, International Journal of Bio-Inspired Computation (IJBIC), vol. 1, no. ½, 2009, [2] (http://inderscience.metapress.com/media/g3t6qnluqp0uc9j3kg0v/contributions/a/4/0/6/a4065612210t6130.pdf)
43. ^ Baudry, Benoit; Franck Fleurey, Jean-Marc Jézéquel, and Yves Le Traon (March–April 2005). "Automatic Test Case Optimization: A Bacteriologic Algorithm" (http://www.irisa.fr/triskell/publis/2005/Baudry05d.pdf) (PDF). *IEEE Software* (IEEE Computer Society) **22** (2): 76–82. doi:10.1109/MS.2005.30 (http://dx.doi.org /10.1109%2FMS.2005.30). Retrieved 9 August 2009.
44. ^ Civicioglu, P. (2012). "Transforming Geocentric Cartesian Coordinates to Geodetic Coordinates by Using Differential Search Algorithm". *Computers &Geosciences* **46**: 229–247. doi:10.1016/j.cageo.2011.12.011 (http://dx.doi.org/10.1016%2Fj.cageo.2011.12.011).
45. ^ Kjellström, G. (December 1991). "On the Efficiency of Gaussian Adaptation". *Journal of Optimization Theory and Applications* **71** (3): 589–597. doi:10.1007/BF00941405 (http://dx.doi.org/10.1007%2FBF00941405).

Bibliography

- Banzhaf, Wolfgang; Nordin, Peter; Keller, Robert; Francone, Frank (1998). *Genetic Programming – An Introduction*. San Francisco, CA: Morgan Kaufmann. ISBN 978-1558605107.
- Bies, Robert R.; Muldoon, Matthew F.; Pollock, Bruce G.; Manuck, Steven; Smith, Gwenn; Sale, Mark E. (2006). "A Genetic Algorithm-Based, Hybrid Machine Learning Approach to Model Selection". *Journal of Pharmacokinetics and Pharmacodynamics* (Netherlands: Springer): 196–221.
- Cha, Sung-Hyuk; Tappert, Charles C. (2009). "A Genetic Algorithm for Constructing Compact Binary Decision Trees" (http://www.jprr.org/index.php/jprr/article/view/44/25). *Journal of Pattern Recognition Research (http://www.jprr.org/index.php/jprr)* **4** (1): 1–13. doi:10.13176/11.44 (http://dx.doi.org/10.13176%2F11.44).
- Fraser, Alex S. (1957). "Simulation of Genetic Systems by Automatic Digital Computers. I. Introduction". *Australian Journal of Biological Sciences* **10**: 484–491.
- Goldberg, David (1989). *Genetic Algorithms in Search, Optimization and Machine Learning*. Reading, MA: Addison-Wesley Professional. ISBN 978-0201157673.
- Goldberg, David (2002). *The Design of Innovation: Lessons from and for Competent Genetic Algorithms*. Norwell, MA: Kluwer Academic Publishers. ISBN 978-1402070983.
- Fogel, David. *Evolutionary Computation: Toward a New Philosophy of Machine Intelligence* (3rd ed.). Piscataway, NJ: IEEE Press. ISBN 978-0471669517.
- Holland, John (1992). *Adaptation in Natural and Artificial Systems*. Cambridge, MA: MIT Press. ISBN 978-0262581110.
- Koza, John (1992). *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. Cambridge, MA: MIT Press. ISBN 978-0262111706.
- Michalewicz, Zbigniew (1996). *Genetic Algorithms + Data Structures = Evolution Programs*. Springer-Verlag. ISBN 978-3540606765.
- Mitchell, Melanie (1996). *An Introduction to Genetic Algorithms*. Cambridge, MA: MIT Press. ISBN 9780585030944.
- Poli, R.; Langdon, W. B.; McPhee, N. F. (2008). *A Field Guide to Genetic Programming*. Lulu.com, freely available from the internet. ISBN 978-1-4092-0073-4.

- Rechenberg, Ingo (1994): Evolutionsstrategie '94, Stuttgart: Fromman-Holzboog.
- Schmitt, Lothar M; Nehaniv, Chrystopher L; Fujii, Robert H (1998), *Linear analysis of genetic algorithms*, Theoretical Computer Science 208: 111–148
- Schmitt, Lothar M (2001), *Theory of Genetic Algorithms*, Theoretical Computer Science 259: 1–61
- Schmitt, Lothar M (2004), *Theory of Genetic Algorithms II: models for genetic operators over the string-tensor representation of populations and convergence to global optima for arbitrary fitness function under scaling*, Theoretical Computer Science 310: 181–231
- Schwefel, Hans-Paul (1974): Numerische Optimierung von Computer-Modellen (PhD thesis). Reprinted by Birkhäuser (1977).
- Vose, Michael (1999). *The Simple Genetic Algorithm: Foundations and Theory*. Cambridge, MA: MIT Press. ISBN 978-0262220583.
- Whitley, Darrell (1994). "A genetic algorithm tutorial". *Statistics and Computing* **4** (2): 65–85. doi:10.1007/BF00175354 (http://dx.doi.org/10.1007%2FBF00175354).
- Hingston, Philip; Barone, Luigi; Michalewicz, Zbigniew (2008). *Design by Evolution: Advances in Evolutionary Design*. Springer. ISBN 978-3540741091.
- Eiben, Agoston; Smith, James (2003). *Introduction to Evolutionary Computing*. Springer. ISBN 978-3540401841.

External links

Resources

- Genetic Algorithms Index (http://www.geneticprogramming.com/ga/index.htm) Provides a list of resources in the genetic algorithms field

Tutorials

- Genetic Algorithms Computer "evolve" in ways that resemble natural selection can solve complex problems even their creators do not fully understand (http://www2.econ.iastate.edu/tesfatsi /holland.gaintro.htm) An excellent introduction to GA by John Holland and with an application to the Prisoner's Dilemma
- An online interactive GA tutorial for a reader to practise or learn how a GA works (http://userweb.eng.gla.ac.uk/yun.li/ga_demo/): Learn step by step or watch global convergence in batch, change the population size, crossover rates/bounds, mutation rates/bounds and selection mechanisms, and add constraints.
- A Genetic Algorithm Tutorial by Darrell Whitley Computer Science Department Colorado State University (http://samizdat.mines.edu/ga_tutorial/ga_tutorial.ps) An excellent tutorial with lots of theory
- "Essentials of Metaheuristics" (http://cs.gmu.edu/~sean/book/metaheuristics/), 2009 (225 p). Free open text by Sean Luke.
- Global Optimization Algorithms – Theory and Application (http://www.it-weise.de/projects/book.pdf)

Categories: Genetic algorithms | Mathematical optimization | Optimization algorithms and methods | Search algorithms | Cybernetics | Digital organisms

---