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.[1] Some examples of GA applications include optimizing decision trees for better performance, solving sudoku puzzles,[2] hyperparameter optimization, causal inference,[3] etc.

Methodology

Optimization problems

In a genetic algorithm, a population of candidate solutions (called individuals, creatures, organisms, 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.[4]

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 (also called *bit set* or *bit string*).[4] 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 or the distribution of the sampling probability tuned to focus in those areas of greater interest.[5]

Selection

Main article: Selection (genetic algorithm)

During each successive generation, a portion of the existing population is selected to reproduce for 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

Main articles: Crossover (genetic algorithm) and Mutation (genetic algorithm)

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[6][7] 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.[*citation needed*]

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. An adequate population size ensures sufficient genetic diversity for the problem at hand, but can lead to a waste of computational resources if set to a value larger than required.

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.[8][9]

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."[10]

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.[11][12] 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.[13][14][15]

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. This makes it extremely difficult to use the technique on problems such as designing an engine, a house or a plane[*citation needed*]. 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 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 stopping 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,[16] although the No Free Lunch theorem[17] 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 binary pass/fail outcome (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

Main article: genetic 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.[18][19]

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.[20] 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.[21]

Parallel implementations

Parallel implementations of genetic algorithms come in two flavors. 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. Researchers have analyzed GA convergence analytically.[22][23]

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),[24] the adjustment of *pc* and *pm* depends on the fitness values of the solutions. There are more examples of AGA variants: Successive zooming method is an early example of improving convergence.[25] In *CAGA* (clustering-based adaptive genetic algorithm),[26] 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. Recent approaches use more abstract variables for deciding *pc* and *pm*. Examples are dominance & co-dominance principles[27] and LIGA (levelized interpolative genetic algorithm), which combines a flexible GA with modified A* search to tackle search space anisotropicity.[28]

It can be quite effective to combine GA with other optimization methods. A 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[*citation needed*] 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.[29]

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,[30] GEMGA[31] and LLGA.[32]

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[*citation needed*]. GAs have also been applied to engineering.[33] 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,[34] antennae designed to pick up radio signals in space,[35] walking methods for computer figures,[36] optimal design of aerodynamic bodies in complex flowfields[37]

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 [sic] 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[38]:267

History

In 1950, Alan Turing proposed a "learning machine" which would parallel the principles of evolution.[39] 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.[40][41] His 1954 publication was not widely noticed. Starting in 1957,[42] 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)[43] and Crosby (1973).[44] 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.[45] Other noteworthy early pioneers include Richard Friedberg, George Friedman, and Michael Conrad. Many early papers are reprinted by Fogel (1998).[46]

Although Barricelli, in work he reported in 1963, had simulated the evolution of ability to play a simple game,[47] artificial evolution only 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.[48][49][50][51] 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.

Commercial products

In the late 1980s, General Electric started selling the world's first genetic algorithm product, a mainframe-based toolkit designed for industrial processes.[52][*circular reference*] 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[53] about Evolver in 1990, and it remained the only interactive commercial genetic algorithm until 1995.[54] Evolver was sold to Palisade in 1997, translated into several languages, and is currently in its 6th version.[55] Since the 1990s, MATLAB has built in three derivative-free optimization heuristic algorithms (simulated annealing, particle swarm optimization, genetic algorithm) and two direct search algorithms (simplex search, pattern search).[56]

Related techniques

See also: List of genetic algorithm applications

Parent fields

Genetic algorithms are a sub-field:

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

Related fields

Evolutionary algorithms

This section **needs additional citations for verification**. Please help improve this article by adding citations to reliable sources in this section. Unsourced material may be challenged and removed. *(May 2011)* *(Learn how and when to remove this template message)*

Main article: Evolutionary algorithm

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.[57] 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.
- Estimation of Distribution Algorithm (EDA) substitutes traditional reproduction operators by model-guided operators. Such models are learned from the population by employing machine learning techniques and represented as Probabilistic Graphical Models, from which new solutions can be sampled[58][59] or generated from guided-crossover.[60]
- 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. There are many variants of Genetic Programming, including Cartesian genetic programming, Gene expression programming,[61] grammatical evolution, Linear genetic programming, Multi expression programming etc.
- 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.[62] 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

Main article: Swarm intelligence

Swarm intelligence is a sub-field of evolutionary computing.

- Ant colony optimization (**ACO**) uses many ants (or agents) equipped with a pheromone model to traverse the solution space and find locally productive areas.
- Although considered an Estimation of distribution algorithm,[63] 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.[64]

Other evolutionary computing algorithms

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

- 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, there is not one individual that fits the whole environment. So, one needs 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.[65]
- 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 evolution (DE) inspired by migration of superorganisms.[66]
- 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.[67] 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 candidate 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 metaheuristics.