

## Chapter 2

# Overview of Current Metaheuristic Paradigms

**Abstract** At this point some of the most visible paradigms related to global optimization using metaheuristics are described. The general intention is to describe well-established methods and show their usefulness in several difficult optimization scenarios. Although there exist a reasonable number of effective algorithms in this class, only a few of them are presented here, in the hope that the selection may give the reader a good idea of the whole context. In principle each one may be used as an “optimization engine” when doing global optimization on manifolds, to be described later.

### 2.1 Introduction

Many problems presenting high complexity can often be expressed as optimization tasks, frequently global optimization ones. Hence, it is usually necessary to define an objective function so that its extreme points with respect to a set of parameters represent the solution to the original question. Frequently the optimization problem is imposed a constraint set [1, 15, 20]. The aim of this chapter is to concentrate on a group of methods, namely metaheuristics, that include paradigms like differential evolution, simulated annealing, genetic algorithms, PSO etc. Having the same common aim of solving hard optimization problems, their working principles are typically inspired by physical, pre-existing systems [2–4, 6–8].

Sometimes the metaheuristics label is not seen as an adequate expression, often used to describe a subfield of stochastic optimization, that is the general class of techniques employing randomness to search for optimal solutions. Metaheuristics could be considered the most general of these types of algorithms, being applied to a very wide range of problems [9].

In this fashion, metaheuristics methods are, to a large extent, based on a common set of principles that make it possible to design powerful algorithms. The large variety of existing metaheuristics could be attributed to different ways of combining those fundamental ideas.

Perhaps due to this fact, it is a not so uncommon mistake to confuse things and say that different paradigms are the same only because they use exploration and exploitation in their search mechanisms.

Of course, almost all stochastic global optimization algorithms do use exploration and exploitation in a probabilistic way, including nature-inspired ones. But this “logic” is wrong, taking into account that the most effective algorithms implement their heuristics in different ways, obviously directed toward maximum efficiency and effectiveness. Anyway, it is advisable to devise ways of classifying metaheuristics.

It is not difficult to find the categories into which metaheuristic methods may be classified. Depending on the chosen criteria, many subdivisions are possible, each of which is the result of a specific perspective. In this chapter the most common ways of classifying metaheuristics are described.

It is common practice to base the classification on the fundamental “inspiration” of the algorithm, as in the division based on whether they are nature-inspired or not nature-inspired ones. Hence, examples of nature-inspired algorithms are particle swarm optimization (PSO) and artificial bee colony (ABC) methods [11, 12, 16], and not nature-inspired ones could be grenade explosion and cross-entropy. Naturally, there are occasions when it is not simple to classify a given paradigm, but it is somewhat usual to find references to this classification scheme in the literature.

A different classification scheme is related to the number of states stored by algorithms during their search processes, namely, population-based and single point search. This scheme is based on the number of candidate solutions managed simultaneously and, as the name indicates, depends on whether the algorithm keeps one or many candidate solutions during the search process. Single solution methods tend to be simpler in terms of data management and almost memoryless, describing a trajectory in the search space during the whole optimization process. Population-based methods, on the other hand, drive multistate processes, evolving many auxiliary components along the whole search period.

Yet another way to classify metaheuristics is to consider the way they use objective functions. Most algorithms use a fixed objective function, while others modify it during the search, trying to escape from local minima by changing the original landscape. Therefore, during the optimization process the objective function can be modified, depending on the information obtained so far.

In the following, the main characteristics of some well-known methods are described.

## 2.2 Particle Swarm Optimization (PSO)

The PSO paradigm belongs to the class of swarm intelligence methods, common approach when solving global optimization problems [4, 16]. Used at first to simulate social behavior scenarios, it was employed as an optimization method in the nineties. Being related to swarm-based theories and evolutionary computing, it is easily implemented in well-known programming languages and demands few computational resources. Furthermore, it requires only objective function values (no derivative information), demonstrating to be an efficient method for complex optimization problems. Its fundamental working principle is based on maintaining

a population of candidate solutions to the problem under study, whereas each individual in the population has variable velocity, making it move through the state space. In addition each particle possess a private memory, recording the best position of the search space that it has ever visited so far in the progressing simulation. In this fashion, the evolution proceeds toward a collective tendency to the best previously visited positions.

Showing coherence with the fundamental working principles of swarm intelligence heuristics, its basic operations are simple and inexpensive in computational terms, being also responsive to changes in the overall environment, not changing its behavior every time the context changes.

It is noticeable that there are differences between PSO and usual evolutionary computing techniques in that the latter feature typically three main operations, namely, recombination, mutation and selection. PSO does not present a strict recombination operation, but the stochastic acceleration of an individual toward its previous best position, as well as in the direction of the best particle of the group, is similar to the recombination procedure in evolutionary processes. Also, the information swap occurs only between the particle's own history and that of the best particle in the set, instead of being carried from fitness dependent selected "parents" to "descendants". Furthermore, the directional position updating operation of PSO could be compared to the mutation of GA.

In summary, PSO happens to be a useful technique for solving complex global optimization problems, although additional research is necessary to completely establish its dynamics and full power.

## 2.3 Differential Evolution (DE)

It is a simple, easy to use, and effective method for global optimization of real-valued, multimodal objective functions [18]. Its implementations are typically efficient, not requiring too much housekeeping, with respect to parameter tuning. In its standard construction, it utilizes just a few control variables which are kept fixed during optimization procedures, but there are many variants of differential evolution with adaptive dynamics. In this optimization model, populations are formed by sets of individuals represented by vectors. New parameter vectors are synthesized by adding the weighted difference between two population vectors to a third vector. When the resulting vector is better than a given individual, a substitution takes place.

Although the method is relatively new, there exist several variants that characterize themselves by the driving strategy they modify/evolve existing vectors, that is to say, it is possible to compose individuals with more than one weighted difference vector or mix the parameters of the old vector with those of the perturbed one before comparisons. The many existing schemes normally differ in the way a new element is created—operations similar to GA's crossover occur after generating new points, by composing certain aspects of involved individuals. To make the actual combination, an offset within the vector and a crossover amplitude are randomly chosen, being the

control parameter for the crossover size a number in the interval  $[0, 1]$ , determining its probability distribution. This crossover-like procedure works by swapping a strip of a certain size from the generated individual and the corresponding part of the current one. The next step is to evaluate the resulting point, replacing the present one in case of improvement.

The basic set of parameters controlling the dynamics of this method consists of population size, “crossover-like” amplitude and a non-null element of  $[-1, 1]$ , the latter controls the degree of influence that differences have on the generation of mutant individuals. After this brief description it is possible to conclude that differential evolution is very easy to deal with and apply, but there is no consensus of which scheme is the most adequate, nor is there a definitive indication of which parameter values to use in each case. Nevertheless, this paradigm keeps improving since its original version and have been applied to several complex problems, demonstrating very good performance in many of them.

## 2.4 Genetic Algorithms

This paradigm was introduced decades ago and is still intensively used today, especially its more recent variations, incorporating increasing degrees of adaptability [10, 14]. The fundamental algorithm is very simple and works by iterating through fitness evaluation (fitness function calculation), selection, mating and population rearrangement. The differences among the several implementations reside, for example, in the way selection and breeding occurs. In addition, new ways of chromosome coding may provoke a significant difference in the final algorithm—typically it is possible to find binary and floating-point types. They deeply differ in the way operations of mutation, reproduction and recombination are constructed. Actually, many meanings are attributed to the term *genetic algorithm*, but a genetic algorithm may be defined as a population-based method using selection, reproduction and mutation operations, aiming to evolve a variable set of candidates in a certain state space toward a point of maximum fitness. Since the initial formulation, several types of genetic operators and chromosome representations were proposed for solving a large number of optimization problems. Being inspired by biological evolution, the encoding of potential solutions uses chromosome-like data structures that are submitted to recombination operations so as to preserve positive features. The initial population is composed of individuals (chromosomes) usually chosen in a randomized way and, from then on, evolution occurs by evaluating elements in each generation and inducing reproduction so that better solutions have greater probability of generating offspring. The classical procedure for evolving a given population begins with an empty set, selects parents from the original population, then copies, combines them with one another, mutating the resulting individuals—this sequence happens until a new population is formed.

As expected, more adequate individuals are given greater opportunities to reproduce and the genetic operators (usually mutation and crossover) are applied to the

individuals in the mating buffer, producing offspring that will compose the next generation. Obviously, mutation and crossover processes can be designed in several ways, in particular the mutation rate is a very influential parameter, considering that if the rates are low, the resulting offspring tend to be very similar to their parents, and vice versa. In a distinct design dimension, it is necessary to decide about how many individuals are to be produced by crossover, and how many are to be selected and paired in the reproduction buffer. As almost all genetic algorithms keep fixed-sized populations, it is necessary to select the given number of individuals resulting from the overall processes described above. It is possible, for instance, to keep all generated children and select individuals from the old population to survive in the new population. In that case, each “birth” in the new population implies in one “death” in the old population, so to speak. In the limit (two new individuals associated to two parents), the previous population completely vanishes, being totally replaced by the new population. If reproduction is based on the fitness degree, the probability of an individual being chosen is mainly based on its objective function value. This strategy may be implemented by scaling fitness values of the parent population and computing the probability of any individual being selected, with basis on the respective values. Although scaling mechanisms are able to attenuate the problem of weak selective pressure, not rarely the use of fitness-based selection may have the opposite effect, that is, increased selective pressure. Therefore, individuals presenting high fitness have greater tendency of being selected. Going further, if many instances of a high fitness individual are placed in the mating buffer, the new population will probably contain many of its clones, provoking premature convergence. To avoid this, fitness proportional selection is replaced with ranked selection, that ranks individuals in the parent population, making the probability of selection a function of rank instead of fitness. Another well-known way of performing choices is tournament selection, in which a reduced subset of individuals is randomly chosen and the best individuals are selected for reproducing. A common type of selective criterion that preserves copies of the best individuals is referred to as elitism—it is an attempt to preserve the current quality, expecting that it could help finding better elements in future generations.

## 2.5 Grenade Explosion Method (GEM)

The underlying idea of the grenade explosion algorithm [2] is (obviously) inspired by the dynamics of a grenade explosion, during which the thrown pieces of shrapnel destroy objects near the location where the explosion takes place. The damage caused by fragments of shrapnel is computed, and high values for loss per piece of shrapnel in an area indicates the existence of valuable goods in that neighborhood. In order to provoke more losses, the next grenade should be directed to the point with greatest losses. Even though objects nearby grenade’s location are more likely to be destroyed, the chance of destruction for distant objects may be amplified by choosing a high value for the parameter representing the length of explosion along each coordinate. By repeating this procedure, it would eventually result in finding the best place for

throwing the grenades. The fitness of the objective function at the object's location is taken as the loss caused by destruction of an object.

One outstanding feature of this paradigm is the agent's territory radius, representing the idea that agents (the same as grenades) do not permit other agents to come closer than a specific distance. Hence, whenever several agents explore the feasible space, higher values for this parameter force grenades to be positioned in a more uniform way in the search domain and the whole space tends to be explored. On the other hand, lower values allow grenades to get closer in order to investigate local regions jointly. Better exploration of distant regions is achieved when higher values for the explosion range are established, whereas lower values tend to drive the algorithm to focus the search on smaller sub-domains, leading to better exploitation.

By comparing results obtained from the optimization of several benchmark functions using GEM [2] and other evolutionary methods, it is evidenced that GEM is able to find global minima locations efficiently. Besides, according to the tests, it is often able to find the global minimum of multimodal functions without being caught inside local minima attraction regions. In summary, the method has shown promising performance but may be improved—this can be achieved, for example, by automatically tuning certain control parameters or pruning the overall control set, in order to reduce the housekeeping effort.

## 2.6 Artificial Bee Colony Algorithm (ABC)

ABC algorithm [11] is population-based and maintains a colony of artificial agents (bees) that can be of three types: employed, onlookers, and scouts. Onlookers are bees waiting in the dance area for making decisions about food sources. Employed bees go to food sources visited by themselves previously. Scouts execute random searches. The first half of the colony is integrated by employed bees and the second half by onlookers, and to each food source corresponds only one employed bee. Employed bees become scouts whenever their food sources get exhausted.

Loosely speaking, after initializing data structures, the basic algorithm works by cycling through three steps:

- Directing employed bees to food sources and measuring the respective nectar amounts;
- Selecting food sources by onlookers after sharing the information of employed bees and determining nectar amount of the foods;
- Choosing the scouts and sending them to potential food sources.

During the initialization stage, a collection of food source positions are randomly chosen by bees and their nectar amounts are calculated. After this, bees get into the hive and share the nectar information with other ones waiting in the dance area, inside the hive. In the course of the second stage, the information is shared and all employed bees go to the food source area visited by themselves in the previous cycle, then

choose new food sources using visual information in the surroundings. In the third phase, onlookers favor food sources areas depending on nectar information spread by the employed bees in the dance area. In this fashion, the probability of choosing a food source is directly proportional to the nectar amount existing in it. So, onlookers for the food source areas with higher nectar amount are selected and, after arriving at the determined area, a new food source is chosen in the neighborhood of the one in the memory depending on visual information, which is based on the comparison of food source positions. Also, whenever the nectar of a food source is abandoned by bees, a new food source is determined by a scout bee and occupies the position of the discarded one. In the original method, at each cycle, only one scout searches for new food sources, and the same number of employed and onlooker bees is used. In the original algorithm, positions of food sources represent possible solutions for the optimization problem, and nectar quantities associated to food sources correspond to quality (fitness) of corresponding solutions. Furthermore, the number of employed bees is the number of solutions and, firstly, ABC generates a random initial population. Each food source is represented by a vector containing the parameters under optimization, and the population of potential solutions suffers successive transformations through many cycles of the search processes of the employed, onlooker, and scout bees. Employed or onlooker bees produce random modifications to solutions in the memory for the sake of finding new food sources and sample the nectar amount of the new source. Similar to what occurs in Nature, where the selection of new food sources is based on comparisons among food sources in a certain region, depending on the available information, in the ABC paradigm the detection of new food sources is also based on comparison of food source positions. Differently from the real ones, artificial bees do not use any information when making comparisons, they just select food source positions at random and produce modifications in their memories. In addition, if the nectar content of the new source is greater than that of the previous one, the new position is stored and the old one forgotten. In the contrary case, the position of the previous one is maintained and, after all employed bees complete the search process, the nectar and position information of food sources is shared with the onlooker bees. Onlookers evaluate nectar information obtained from employed bees and choose food sources in a random way, but related to their content. As in the case of employed bees, a change in positions is produced in memory and the nectar amounts of candidates are checked. If the nectar quantity is higher than that of the previous one, the new position is kept and the old one forgotten. Considering that in robust search processes exploration and exploitation mechanisms are very important and complementary, in ABC, onlooker and employed bees are responsible for executing the exploitation task, while scouts control the exploration of the search domain.

In summary, in [11] ABC is compared to GA, PSO, and PS-EA, which are swarm and population based approaches as well . The tests demonstrated good performance of ABC algorithm when optimizing five high dimensional cost functions featuring multimodal landscapes, and the conclusion is that it is able to escape from local minima and can be efficiently used for multivariable, multimodal function optimization.



## 2.7 Cross-Entropy Method (CE)

This algorithm was introduced in 1997, aimed at estimating probabilities of rare events by minimization of sample variance of the importance sampling estimator. Later modified to minimize the Kullback-Leibler divergence, evolved into the present cross-entropy method [17]. The CE method can be easily applied to the solution of continuous multimodal optimization problems [17] and may be described as an iterative algorithm which consists of repeating two steps in each execution cycle, namely, generation of random data samples using a set of dynamic parameters and updating the current set of parameters controlling the generation of random data using the current sample, trying to improve data in the next iteration.

The cross-entropy method presents a versatile structure, easily adjustable to a wide range of applications, with a very simple and efficient updating procedure. The measurements (individuals) are obtained from simulations as a random vector  $X = (X_1, \dots, X_n)$  having probability density function  $f$ . As in other types of evolutionary algorithms, information about the problem is contained in a population of potential solutions. In CE, the information about the experiment is concentrated in the current pdf, narrowing down the search for information by compressing it into a parametric vector. The accuracy with which the true parameter vector can be estimated from a result of a given simulation is related to the amount of information about it which is contained in the present sample  $X$ .

Another important feature is the choice of a method for measuring the amount of information by using the concept of distance which, in CE, is the Kullback-Leibler divergence. On the other hand, the measurement of information may be done by using the concept of Shannon entropy, which defines the average number of bits required to send a message  $X$  through a communication channel.

As mentioned above, the paradigm has two phases, which are repeated until a pre-defined stopping criterion is met: after producing the initial sample according to a specified probabilistic mechanism, the relevant parameters are modified with basis on the current population. Following this, a new sampling phase starts (using the newest estimated parameters) in order to obtain a better estimative of the true parameters, corresponding to the actual population. This process repeats until the solution is obtained or the maximum number of iterations is reached.

Therefore, the CE method may be considered as a kind of search process based on importance sampling distributions that approximates the true distribution corresponding to the objective function under minimization and, to reach that aim, Kullback-Leibler divergence is taken as a kind of proximity measure, guiding the overall approximation task. Hence, it can be viewed as a minimization problem whose search space is a class of probability distribution functions, and the objective function is proportional to the difference between candidate probability distribution functions and the function describing the actual landscape.



## 2.8 Simulated Annealing

Initially designed to handle combinatorial optimization problems [13, 19], this method has been successful in high-dimensional domain problems, being based on evaluations of objective functions, and allowing transitions out of sub-optimal attraction basins. Although there is no guarantee of finding global minima, the algorithm tends to approach them in a probabilistic sense. It is also able to discriminate between large oscillations of the landscape and small perturbations. The guiding principle is, at first, to exploit objective function's domain and reach areas in which at least one global minimum must be present. At this stage, it tries to find good, near-optimal local minima, or perhaps a global minimum. In [5] some modifications to this algorithm are made in order to use it in the optimization of functions defined on continuous domains—these functions do not need to be smooth or continuous in their domain, and the algorithm works based exclusively on objective function evaluations. The method assumes that  $f : C \rightarrow \mathbb{R}^n$  is the bounded function to minimize, where  $C$  is a hyper-rectangle and  $f$  does not need to be continuous. It starts from a given point  $x_0$  and generates a succession of points  $x_1, x_2, \dots$  approaching the global minimum of the cost function  $f$ . During each new iteration, candidate points are generated around the current point  $x_i$  by perturbing it along each coordinate direction.

The SA algorithm starts at a user defined temperature  $T_0$  and generates a sequence of points until statistical mixing occurs. Next, the temperature  $T$  is lowered and a new sequence of moves is made, starting from the best point resulting from the previous phase, until a new equilibrium takes place again. The process is stopped at a temperature such that no more significant improvement could be reached, satisfying a pre-established criterion.

The simulated annealing framework may be viewed as a method of simulation of physical processes in which metallic objects are driven to states of minimum energy. In Nature this is possible by cooling metals slowly, until they get to a highly ordered state of low energy levels. On the other extreme, fast cooling schedules provoke a kind of disorder inside the material, making it harder after reaching lower temperatures. This behavior is compatible with the experimental fact that a search process that only accepts new points with lower cost function values tends to get stuck in local minima attraction basins. In this fashion, due to its decision criterion it permits uphill moves. When at higher temperatures, only the global behavior of the cost function is relevant to the search dynamics and, as temperature values decrease, smaller neighborhoods can be explored, allowing the algorithm to reach more precise results. Not existing deterministic guarantees of reaching global minima, there is a statistical proof of convergence and, in practice, the method is typically able to proceed toward global minima even in the presence of multimodal landscapes.

Although it is expensive in terms of number of function evaluations, it is capable of finding global minima of difficult objective functions and provides a very high reliability in the minimization of multimodal functions. In another direction, some design decisions have to be addressed as, for example, the choice of starting temperatures. By choosing a very high one, there will be a waste of computational resources.

In the opposite extreme, it is very likely to be caught in sub-optimal neighborhoods. In addition, objective functions' landscapes vary to such an extent that it is very hard to establish general rules for calculating good starting temperatures and, in practice, some experimentation may be of great help in finding adequate values for it and other significant control parameters.

While at high temperatures, practically any change is accepted—the algorithm sweeps a relatively large territory around the current state. On the other hand, at lower temperatures, transitions to higher energy states become less frequent and the search tends to concentrate in smaller regions. The cooling schedule is very important to the performance of simulated annealing approach, consisting of initial temperature, decrement function for the temperature, final temperature determined by a stopping criterion, and the number of transitions in the homogeneous Markov chain at each temperature stage. As expected, the quality of the final solution is directly proportional to the total execution time, the latter being dependent on the decrement speed of the temperature.

## References

1. Aguiar e Oliveira Jr, H., Ingber, L., Petraglia, A., Petraglia, M.R., Machado, M.A.S.: *Stochastic Global Optimization and Its Applications with Fuzzy Adaptive Simulated Annealing*. Springer, Berlin (2012)
2. Ahrari, A., Atai, A.A.: Grenade explosion method—a novel tool for optimization of multimodal functions. *Appl. Soft Comput.* **10**, 1132–1140 (2010)
3. Birbil, S.I., Fang, S.: An electromagnetism-like mechanism for global optimization. *J. Glob. Optim.* **25**, 263–282 (2003)
4. Clerc, M.: *Particle Swarm Optimization*. ISTE Publishing Company, London (2006)
5. Corana, A., Marchesi, M., Martini, C., Ridella, S.: Minimizing multimodal functions of continuous variables with the simulated annealing algorithm. *ACM Trans. Math. Softw.* **13**, 262–280 (1987)
6. Deb, K.: *Multi-Objective Optimization Using Evolutionary Algorithms*. Wiley, New York (2001)
7. Dorigo, M., Stützle, T.: *Ant Colony Optimization*. Bradford Books, Cambridge (2004)
8. Dréo, J., Pétrowski, A., Siarry, P., Taillard, E.: *Metaheuristics for Hard Optimization Methods and Case Studies—Simulated Annealing, Tabu Search, Evolutionary and Genetic Algorithms, Ant Colonies*. Springer, Berlin (2006)
9. Glover, F., Kochenberger, G.A.: *Handbook of Metaheuristics*. Springer, Boston (2003)
10. Holland, J.H.: *Adaptation in Natural and Artificial Systems*. University of Michigan Press, Ann Arbor (1975)
11. Karaboga, D., Basturk, B.: A powerful and efficient algorithm for numerical function optimization: artificial bee colony (ABC) algorithm. *J. Glob. Optim.* **39**, 459–471 (2007)
12. Karaboga, D., Basturk, B.: On the performance of artificial bee colony (ABC) algorithm. *Appl. Soft Comput.* **8**, 687–697 (2008)
13. Kirkpatrick, S., Gelatt Jr, C.D., Vecchi, M.P.: Optimization by simulated annealing. *Science* **220**, 671–680 (1983)
14. Michalewicz, Z.: *Genetic Algorithms + Data Structures = Evolution Programs*. Springer, Heidelberg (1994)
15. Nocedal, J., Wright, S.J.: *Numerical Optimization*. Springer, New York (1999)

16. Parsopoulos, K.E., Vrahatis, M.N.: Recent approaches to global optimization problems through particle swarm optimization. *Nat. Comput.* **1**, 235–306 (2002)
17. Rubinstein, R.Y., Kroese, D.P.: *The Cross-entropy Method: A Unified Approach to Combinatorial Optimization, Monte-Carlo Simulation, and Machine Learning*. Springer, New York (2004)
18. Storn, R., Price, K.: Differential evolution—a simple and efficient heuristic for global optimization over continuous spaces. *J. Glob. Optim.* **11**(4), 341–359 (1997)
19. van Laarhoven, P.J.M., Aarts, E.H.L.: *Simulated Annealing: Theory and Applications*. D. Reidel, Dordrecht (1987)
20. Weise, T.: *Global Optimization Algorithms—Theory and Application*. Available as e-book at <http://www.it-weise.de/>. Accessed 11 July 2014

Evolutionary Global Optimization, Manifolds and  
Applications

Aguiar e Oliveira Junior, H.

2016, XIII, 137 p. 43 illus., 26 illus. in color., Hardcover

ISBN: 978-3-319-26466-0