International Journal of Computational Intelligence Systems, Vol. 8, No. 4 (2015) 606-636

REVIEW ARTICLE

A Walk into Metaheuristics for Engineering Optimization: Principles, Methods and Recent Trends

Ning Xiong
School of Innovation, Design and Engineering, Mälardalen University
Västerås, SE-72123, Sweden
E-mail: ning.xiong@mdh.se

Daniel Molina
Department of Computer Science and Engineering, University of Cadiz
Cadiz, 11001, Spain
E-mail: daniel.molina@uca.es

Miguel Leon Ortiz
School of Innovation, Design and Engineering, Mälardalen University
Västerås, SE-72123, Sweden
E-mail: miguel.leonortiz@mdh.se

Francisco Herrera
Department of Computer Science and Artificial Intelligence, University of Granada,
Granada, 18071, Spain
School of Innovation, Design and Engineering, Mälardalen University
Västerås, SE -72123
E-mail: herrera@decsai.ugr.es

Received 5 October 2014

Accepted 15 December 2014

Abstract

Metaheuristics has attained increasing interest for solving complex real-world problems. This paper studies the principles and the state-of-the-art of metaheuristic methods for engineering optimization. Both the classic and emerging approaches to optimization using metaheuristics are reviewed and analyzed. All the methods are discussed in three basic types: trajectory-based, in which in each step a new solution is created from the previous one; multi-trajectory-based, in which a multi-start mechanism is used; and population-based, where multiple new solutions are created considering a population of approximate solutions. We further discuss algorithms and strategies to handle multi-modal and multi-objective optimization tasks as well as methods for parallel implementation of metaheuristic algorithms. Then, different software frameworks for metaheuristics are introduced. Finally, several interesting directions are pointed out as future research trends.

Keywords: metaheuristics, optimization methods, trajectory-based optimization, population-based optimization, multimodal optimization, multi-objective optimization, parallel metaheuristics

1. Introduction

Nowadays, optimization has become an important issue in industrial systems design and product development [1]. It is necessary to enhance system performance whereas reduce product cost to meet challenges in the competitive market. From engineering perspective, optimization is concerned with adjusting or fine tuning system designs in terms of one or more performance factors. This is not a trivial task particularly when the problem space is complex and with high-dimensionality.

Generally, optimization techniques can be divided into two categories: linear programming [2] and non-linear programming [3], [4]. The former is applied to optimization problems that have linear objective and constraint functions. The method for linear programming was first invented in 1947 with the use of simplex to solve linear programs. Further important progresses in this area include the polynomial-time ellipsoid algorithm [5] and the interior point algorithm [6], both were proposed to reduce time complexity and to allow for extremely efficient problem handling in the optimization procedure. At present, linear programming has been advanced to a soundly founded discipline and widely used technology for linear optimization problems. The second category of optimization is called nonlinear programming, which refers to the consortium of methods and approaches that are designed to deal with problems with nonlinear objective or constraint functions [7]. Nonlinearity is a very common property for many engineering optimization problems, and solving such problems often presents a challenge due to high complexity, high dimensionality and multi-modality of the problem space.

Metaheuristics [8] has been developed to tackle nonlinear, complex optimization problems for which exact optimization techniques fail to offer satisfactory results. It is implemented through an iterative generation process that guides subordinate heuristics in exploration and exploitation of the search space to efficiently find near-optimal solutions [9]. Metaheuristic algorithms are not problem and domain specific, and they are capable of locating good quality solutions in a relatively shorter time, compared to traditional optimization techniques .

This paper focuses on the ideas and principles of metaheuristic approaches to tackling hard (nonlinear) optimization problems. We classify existing metaheuristic optimization techniques into three types: trajectory-based approaches, multi-trajectory based approaches and population-based approaches. Significant methods of these types are reviewed and analyzed respectively. We further discuss strategies and algorithms for multi-modal and multi-objective tasks as well as methods for parallel implementation of metaheuristic algorithms, which represent important issues for application of metaheuristics in solving many engineering problems. Some interesting directions of further study are also highlighted as future research trends. Additionally, several relevant issues of interest in the literature are discussed due to their interest, such us, the balance between exploration and exploitation, the influence of the non-free lunch problem, …

Our paper is different to other general reviews in the literature as [10] because it has a different perspective: it not only introduces metaheuristics in a more concise way making references to different specific reviews when they are available, but it also has specific sections about the research areas and relevant topics that are considered specially interesting (at present and in a near future).

The organization of this paper is as follows. Section 2 highlights the basic idea and principle for solving optimization problems. The reviews of three types of metaheuristic methods: **trajectory-based optimization, multi-trajectory based optimization,** and **population-based optimization** are given in Sections 3, 4 and 5 respectively. Section 6 is devoted to discussing three current issues (multi-modal and multi-objective optimization, parallel metaheuristics) which are important for applying metaheuristics in engineering practices. Section 7 makes an overview of some metaheuristic optimization frameworks as software tools. Some future trends of research are highlighted in Section 8. Section 9 introduces several relevant discussion topics in the literature. Finally, conclusion is given in Section 10.

2. General Principle of Optimization

Generally an engineering optimization problem can be formulated as

$$\text{Minimize} \quad f\left(x_1, x_2, \cdots, x_n\right)$$

$$\text{Subject to} \quad \left(x_1, x_2, \cdots, x_n\right) \in \Omega$$

where (x_1, x_2, …, x_n) is the vector of design variables, Ω denotes the region of feasible solutions in the decision (search) space, and $f(X)$ is an objective function providing numerical assessment for vectors of variables representing alternative solutions. The design variables x_i can take continuous or discrete values or a mixture of both depending on problems of interest.

Mathematically it is well known that an optimum of a (nonlinear) objective function $f(x_1,x_2,...,x_n)$ must be some point at which the partial derivatives of the function with respect to all variables are equal to zero, i.e.,

$$\frac{\partial f}{\partial x_i}=0, \quad i=1,2,\cdots,n \tag{1}$$

A solution satisfying all the equations in (1) is called a stationary point of function f. Further the stationary point is a minimum solution if the Hessian matrix of the second-order derivatives, as defined in Eq. (2), is positive definite.

$$H=\begin{bmatrix} \frac{\partial^2 f}{\partial x_1^2} & \frac{\partial^2 f}{\partial x_1 \partial x_2} & \cdots & \frac{\partial^2 f}{\partial x_1 \partial x_n} \\ \cdots & \cdots & \cdots & \cdots \\ \frac{\partial^2 f}{\partial x_n \partial x_1} & \frac{\partial^2 f}{\partial x_n \partial x_2} & \cdots & \frac{\partial^2 f}{\partial x_n^2} \end{bmatrix} \tag{2}$$

The above principle suggests a simple procedure to obtain exact solutions of an optimization problem. It is done by finding all stationary points of the objective function and then examining the property of the Hessian matrices of these points. The global optimum solution is selected from those stationary points for which the Hessian matrices are positive definite.

Unfortunately, the approach to exact solutions of optimization is rarely applicable in engineering practice. The main reason lies in the difficulty of acquiring the derivative information analytically. In many applications, only concrete objective values of individual designs are calculable through specific calculations such as simulation. But the explicit expression of the objective function is not available, not to mention the analytic formulation of the partial derivative functions. It follows that we are unable to construct the equations as formulated in Eq. (1) for determining the stationary points of the objective function.

Metaheuristic methods present a pragmatic alternative to solve engineering optimization problems. The main idea is to create arbitrary initial approximate(s) to the problem and then improve them progressively. The whole procedure consists of a number of iteration steps. In each step of the iteration, new approximate(s) are created from the old one(s) as more promising solution(s). Depending on how the approximates are generated at a single step, three types of metaheuristic methods (trajectory-based, multi-trajectory based and population-based) can be defined and explained as follows.

With trajectory-based approaches, one point as new approximate is generated and maintained in each iteration. The new point is made as transition from an old one with expected better performance. The general form of such transition can be expressed as

$$X_{i+1}=X_i+h_iS_i \tag{3}$$

where X_{i+1} and X_i denote the new and old approximates respectively, h_i decides the length of transition, and S_i is a vector determining the direction of the move from X_i. There are many different methods to adapt h_i and to determine the direction vector S_i in the literature. Some uses merely values of the objective function while others require partial derivative information in addition to the objective values.

Because in trajectory-based approaches, a new solution is always obtained from the previous one, the selection of the initial point (usually randomly generated) has a strong influence over the search. In case of an improper initial point, the algorithm could get stuck in a local optimum, not finding the global optimum. Multi-trajectory based algorithms try to reduce that strong dependency by the incorporation of a multi-start mechanism that repeats the search again from a different initial point when the search is stuck in a local optimum.

Population-based approaches start from an initial population of feasible solutions. Then it undergoes an iterative procedure in which the population evolves to reach progressively refined approximates to the

optimum. This evolution is carried out by the creation of new solutions that are introduced into the population replacing previous solutions, or by the adaptation of existing solutions. As many points in the space are explored simultaneously, population-based approaches are superior to trajectory-based approaches in the global search ability; hence it has less likelihood of ending with a local optimum. Many nature inspired optimization techniques rely on transitions of populations, such as genetic algorithms, genetic programming, evolutionary strategy, evolutionary programming, estimation of distribution algorithms, particle swarm optimization, differential evolution, artificial bee colony algorithms, memetic algorithms, ant colony optimization, as well as scatter search, which will be reviewed in Section 5.

3. Trajectory-Based Optimization

The approaches of this type explore the problem space via transition from one feasible solution to another (although in constraints optimization sometimes non-feasible solutions are used during the search). The transition procedure is controlled following some rules.
The majority of these techniques starts by creating an arbitrary solution (approximate) to the problem and it then create a new solution. If the next generated solution has a better objective value than the current solution, the current one is replaced by that solution and the search moves on to the next iteration. These techniques are particularly recommended when there is limited time for search, for example for real-time systems.

Six well-known approaches in this class will be surveyed here.

3.1. *Hill-Climbing*

Hill-climbing [11] is the simplest metaheuristic approach for optimization. In each step there are generated and evaluated the neighbors of the current solution. The best neighbor replaces the current solution if the neighbor has a better objective value, and the search continues. Hill-climbing is a local search and it is mainly applied in discrete spaces as it implicitly assumes a finite number of feasible neighbors at every point. For continuous optimization, only a small number of neighbors can be generated, thus this technique could not found a local optimum.

The advantages of hill-climbing lie in its simplicity and high efficiency. It has been widely used to solve many machine learning and technical optimization problems (e.g. [12], [13]).

3.2. *Simulated Annealing*

Simulated annealing [14] [15] is a stochastic and metaheuristic algorithm for solving optimization problems, especially focused on avoid get stuck in local optimization. It is inspired by the physical principle of annealing used in material engineering. In an annealing process the solid is first heated to a high temperature, causing atoms to move away from their initial positions. When the material cools down slowly, the atoms adjust themselves into a new thermal equilibrium that corresponds to a minimum energy state.

The algorithm is iterative and the main idea is to randomly select a new solution in the neighborhood of the current solution at every step. The difference of objective values, ΔE, between the new and current solutions is calculated as analogy to the change of energy. If the new solution is better than the current one ($\Delta E<0$), the current solution is replaced by the new one. In case when the new solution is worse ($\Delta E\geq 0$), there is still a chance to move to it. The probability of this move is given by the Boltzmann probability function:

$$P(\Delta E) = \exp\left(-\frac{\Delta E}{T}\right) \tag{4}$$

The parameter T in Eq. (4) is the temperature used during the search. In the early iterations, the temperature is high, which results in high probability of moving into inferior points and thereby avoiding local minima. Contrarily, during late stages, the temperature is reduced to such a level that gives little chance for accepting worse solutions and consequently the search will finally converge to a solution that is optimal, at least locally.

The merit with Simulated Annealing is that it does not require the objective function to be continuous and differentiable, and it can handle both continuous and discrete optimization problems. However, setting adequate parameter values with this method can be difficult.

3.3. *Tabu Search*

Tabu search [16] [17] is a metaheuristic local search algorithm to solve optimization problems, mainly for

discrete optimization but it has been also used for continuous optimization [18]. It can be considered as extension of the hill-climbing search in two aspects as follows. First there is added driving force to enforce the local minimization procedure out of a local minimum. Secondly various memory structures are used to store historical information which is then utilized to guide the further exploration of new solutions. For instance, the tabu list is introduced as short term memory to save recently transformation over the current solution to prevent cyclic behaviors during the search. Intermediate and long term memory is used to intensify and diversify the search to ensure adequate exploration of the problem space.

The search starts from an arbitrary point as the current solution. All the solutions in its neighborhood that are not in the tabu list or satisfy the aspiration level are successor solutions and their objective values are calculated. Then we take the move to the best successor according to the objective values, and the tabu list is updated accordingly. Both uphill and downhill moves are allowed here to give chance to escape from a local minimum. This process is repeated in a number of iterations until the termination condition is satisfied.

It is useful to apply tabu search in many practical scenarios [19] [20], mainly in combinatorial problems. A main limitation with this technique is that it requires domain specific knowledge to design suitable aspiration criteria.

3.4. *Gradient Descent*

Gradient descent [21] aims to solve continuous optimization problems. In this technique, in each step only one solution is generated, and the gradient information is used to identify the direction of move to reduce quickly the value of the objective function. The length of move can be determined by solving a one-dimensional optimization problem. The golden section method is often used in gradient descent to find the optimal size of transition at each step.

Gradient descent is simple and very useful in solving many optimization problems when partial derivatives of the object function are available. However, as local search scheme, this method cannot guarantee the global optimality of the solutions returned. It should preferably be combined with a multi-start strategy to increase the chance of finding the global minimum. The other weakness with gradient descent is that, when the current solution gets close to a minimum solution, the search will become quite inefficient due to the decreasing lengths of the moves.

3.5. Newton's Method

Newton's method [22] attempts to improve the speed of convergence of gradient descent in the vicinity of a minimum solution. According to Taylor's expansion, the objective function near a minimum X^* can be expressed by an approximate form as:

$$f(X) = f(X^*) + \frac{1}{2}(\Delta X)^T H(\Delta X) \tag{5}$$

where H is the Hessian matrix of the second-order partial derivatives of function f. Eq. (5) also reveals that the objective function is approximately quadratic in the vicinity of X^*. It follows that we can obtain the minimum solution X^* from a nearby point X in terms of the following transition rule:

$$X^* = X - H^{-1} g \tag{6}$$

where g is the vector of partial derivatives evaluated at the current point, and H is the Hessian matrix which is constant for a quadratic function.

This transition rule indicates that a single move suffices to reach the minimum X^* when the current solution is nearby. This shows a substantial improvement of the convergence speed compared with that of gradient descent.

Nevertheless it bears noting that globally the objective function is not quadratic. Hence we need an iterative procedure to generate a sequence of moves for progressive refinement. At iteration i, we first calculate g and H at the current point X_i and then we use the Newton's method to create the next refined solution as

$$X^* = X - H^{-1} g \tag{7}$$

Generally, the Newton's method stated above is still a local approach and it has two drawbacks. First it requires heavy computation with the Hessian matrix of second-order derivatives and its inverse. Secondly the method is only efficient in the neighborhood of an optimum, but

away from the optimum it may progress very slowly and even diverge. So our suggestion is not employing the Newton's method alone, but in combination with some other optimization technique and using it at the final stage.

Quasi-Newton method [23] functions as a variant approach that avoids direct calculation of the second order derivatives of the objective function. Instead it builds the Hessian matrix and its inverse successively by analyzing solutions and gradient vectors in the consecutive time steps. The merit of doing this is partially alleviating the heavy computation burden with the Newton's approach. However, Quasi-Newton method has one disadvantage with its behavior in high-dimensional problems. Not only larger data matrices have to be stored and manipulated, but also the error in the estimation increases with the dimensionality.

3.6 Simplex Descent

A simplex is a geometric object consisting of n+1 points (vertices) in the n-dimensional spaces. Every vertex of the simplex corresponds to a feasible solution to the problem. The initial simplex can be generated randomly. The main idea is to move the simplex iteratively and every time replacing the worst vertex of it with a new better point.

According to the simplex method by Nelder and Mead [24], the new better points for replacement are generated through the operations such as reflection, expansion, and contraction. The worst vertex is first reflected through the centroid of the remaining points of the simplex. If the reflection produces a better point, expansion is done to see whether the objective function can be reduced further via moving in the same direction. Otherwise, if the reflected point is not satisfactory, we do contraction by generating a point between the worst vertex and the centroid for possible replacement.

The main advantage of the simplex method is that it is a optimization technique that does not require any derivative information of the objective function. The method is robust and efficient with a small number of design variables but it does not scale well up to problems with greater dimensionality. As noted in [7], the efficiency of simplex diminishes when the design variables are more than five.

4. Multi-Trajectory Based Optimization

All the previous metaheuristics start by creating an arbitrary initial solution. In function of that initial solution, the algorithm could be stuck into one local optimum or another (although simulated annealing sometimes can avoid local optima). Thus, the selection of that initial solution has a great influence over the results obtained. The other possibility is to follow the multi-start strategy [25] when doing optimization with trajectory-based approaches. This means that we run the optimization algorithm multiple times and every time using a different starting point. The best solution found in all the runs is treated as the global optimum.

Four well-known approaches in this class will be surveyed here.

4.1. *Iterative Local Search*

Multi-trajectory optimization has been handled to find different promising solutions as starting points for local search. Iterated local search (ILS) [26] aims to perform a sequence of local searches using different starting points. It generally works in two successive steps: 1) perturbing the current local optimum and 2) applying local search to the modified solution from step 1.

The magnitude of perturbation applied to a current local optimum, perturbation strength, is crucial for the success of ILS. On one hand, the perturbation has to be strong enough to enable the next round of local search to start from a new attraction basin yielding a new local optimal solution. On the other hand, too strong perturbation could imply totally random restart of searches.

4.2. *Variable Neighborhood Search*

Variable neighborhood search (VNS) [27] is similar to ILS in the general structure, yet it allows for systematic change of neighborhood combined with local search applications. Not only the size of neighborhood but also its structure can be adjusted when new solutions are created with respect to a current local optimum in the next iterations.

VNS provides a useful framework to solve both combinatorial and real-coded optimization problems. It has received much research attention and applied in many fields since its inception. Readers are referred to the paper [28] for a thorough review of the basic VNS

algorithm and its extensions as well as the recent applications.

4.3. *Greedy Randomized Adaptive Search Procedure (GRASP)*

Greedy randomized adaptive search procedure (GRASP) [29], [30] is a multi-start and iterative process for solving combinatorial problems. Each iteration in GRASP consists of two successive phases: construction and local search. The first phase aims to build a feasible solution to the problem via an adaptive randomized greedy function. In the second phase, exploration is performed in the neighborhood of the solution constructed in the first phase in order to find a possible improvement. The best solution acquired over all GRASP iterations is kept as the final result.

Solution construction is performed incrementally by incorporating one element to the partial solution at a time. The new element to be added to the partial solution is selected at random from the Restricted Candidate List (RCL), which contains a set of best candidates from the list of all feasible elements. The cardinality of RCL is determined by a parameter which represents a trade-off between greediness and randomness in selecting elements to build a feasible solution. Self-tuning of this parameter was implemented in the Reactive GRASP algorithm [31] for creating opportunity to find better solutions than the basic GRASP method.

4.4. Iterative Greedy

Iterated Greedy (IG) [32], [33] is a metaheuristic that generates a sequence of solutions by iterating over a greedy heuristic using two phases: destruction and construction. The destruction phase removes some solution component from a previous complete solution. The construction phase creates a new solution by applying a greedy constructive heuristic to the result of previous phase. One a candidate solution is generated, an acceptance criterion is applied to decide if the new solution should replace the previous solution.

IG is very closely related to ILS, but there is an important difference: ILS applies LS to perturb the current point to escape from local optima whereas in IG the perturbation is stronger because it is done by a destruction and reconstruction of the solution.

IG has been successfully a many combinatorial problems, like TSP [25], maximum diversity problem [35], and scheduling, including multiobjective scheduling [36] and large-scale scheduling [37].

5. Population-Based Optimization

The approaches of this type explore the problem space via the evolution or replacement of one population of solutions. They are nature inspired techniques and probabilistic rules are used to create new solutions from old ones (or to modify the current solutions). Eleven well known and consolidated approaches in this category will be reviewed here.

However, there are other metaheuristics that are not yet consolidated and hence will not receive careful review in this section. The majority of them are inspired from nature, in different aspects. The examples are: *gravitational algorithms* [38], [39], [40] inspired from the solar system; *harmony search* [41] inspired from music; *bacterial foraging optimization* [42] and *artificial immune systems* [43] inspired from medical concepts; *Biogeography-Based Optimization Algorithms* [44] inspired from migration of species; *variable mesh optimization algorithm* [45] inspired from topology; *glowworm algorithms* [46] inspired from insects; cuckoo search algorithm [47] inspired in the behavior of laying their eggs in the nests of other species of as well as *frog-leaping algorithm* [48] inspired from the frog behavior.

5.1 *Genetic Algorithms*

Genetic algorithms (GAs) are stochastic optimization algorithms that emulate the mechanics of natural evolution [49] [50]. They are attractive to be applied in engineering optimization tasks due to the two following reasons. First, a GA evaluates many points in the search space simultaneously, as opposed to a single point, thus reducing the chance of converging to the local optimum. Second, a GA uses only values of objective functions; therefore they do not require the search space to be differentiable or continuous.

Essentially, a GA is an iterative procedure maintaining a constant population size. An individual in the population is a possible solution to the problem with a string analogous to a chromosome in nature. At each step of iteration, new strings are created via applying genetic operators on selected parents, and subsequently some old weak strings are replaced by new strong ones. In this

manner, the performance of the population will be gradually improved in the evolutionary process.

There are two types of GAs: generational GAs, in which the new population generated (by offspring and selected individuals) replace completely the previous one, and steady-state GAs, in which in each step the offspring can replace existing individuals into the population.

A classical GA works with binary code, i.e., individuals in the population are represented by binary strings. However, binary coding would not be the most appropriate choice in applications to optimization problems with continuous spaces. One reason lies in the matter of resolution, i.e., a binary string is inherently related to some loss of precision for representing the continuous value of a variable. The other reason is the extra job of decoding that is needed when doing fitness evaluation for a binary string in the population.

The other alternative is to directly adopt arrays of real numbers as population individuals. Real-coded GAs have been studied by many researchers and nowadays become a popular, extended version of GAs for solving real-valued optimization problems. The interesting features of real-coded GAs together with their used mechanisms and genetic operators were discussed in [51].

5.2 *Genetic Programming*

Genetic programming (GP) provides a method to automatically create a computer program from high level statements of the problem. It was first proposed by Koza in his book [52] in 1992. Since then GP has attracted much research interest and it has been found very useful for knowledge elicitation in machine learning as well as for solving many real-world problems [53], 54].

The main idea of GP is to evolve a population of computer programs by utilizing the principle of Darwinian natural selection and biologically inspired operators. Hence the general search strategy of GP is very similar to that of GA. However, GP uses flexible structured trees rather than strings to represent individual computer programs in the population. An internal node in the tree is associated with an operator or function, while a leaf node represents a constant or variable.

GP starts with an initial population of randomly created trees. It is important to generate these initial trees with a uniform distribution to cover different sizes and shapes. Then the trees in the population are stochastically selected based on their fitness values for undergoing genetic operators such as reproduction, crossover and mutation. Crossover produces offspring trees by combining subtrees from the parents. Mutation is implemented by a random change of a randomly selected part of the tree. A survey of various crossover and mutation operators designed for GP is given in [55], [56].

5.3 *Evolution Strategies*

Evolution Strategies (ES) were proposed by Rechenberg [57] and further developed by Schwefel. In these algorithms, a generation of offspring are created by selection of solutions (and recombination if more than one parent are involved) and a normally (Gaussian) distributed mutation. In the original model (μ+1)-ES, two parents selected from the μ solutions are combined and mutated, and the new offspring replaces the worse parent if it improves that parent. The other model is (μ+λ)-ES, in which in each step λ solutions are generated and compete with the parents. Another different algorithm is (μ, λ)-ES, with $\lambda > \mu$, in which in each generation λ offspring are generated, and the μ best of them replace the parents, no matter how good or bad the parents are.

One of the most popular and advanced ES models is *Covariance Matrix Adaptation Evolution Strategy*, CMA-ES [58]. CMA-ES is an algorithm that has proven to be very effective in continuous optimization. In this algorithm, the exploration is done by a Gaussian formula that is adapted during the search. In each step, λ solutions are created according to a Gaussian function $N(m, C)$ with mean m and covariance C. These λ individuals are evaluated and ranked, creating the index *rank(i)={i-th best solution}*, and the μ best solutions are used for guiding the search: m is calculated by an average with weights (more influence from solution with better fitness) and C is also adapted, in shape and size, to enforce the generation of solutions similar to the μ best solutions. In [58] these transformations are described in detail.

This CMA-ES algorithm can guide the search very quickly to optimum by its adaptation of parameters, and it is very invariant to transformations. Unfortunately, its results strongly depend on the initial parameters (mainly the initial center of the Gaussian distribution), and it is not very adequate for high-dimension problems [59]. To avoid the dependency on the initial solution, several multi-start algorithms have been proposed that have won

the real coding competitions in the IEEE Congress on Evolutionary Computation: IPOP-CMAES [60], winner in 2005, that increases the population size after each restart; BIPOP-CMA-ES [61], winner in 2009, that combines two different mechanisms of restarts, one with an increasing population size, and the other with a small population size; and an hybrid algorithm, ICMAES-ILS [62] combining an ILS with IPOP-CMA-ES, has won the competition in 2013.

5.4 *Evolutionary Programming*

Evolutionary programming (EP) was originally proposed by Fogel as an evolutionary approach to artificial intelligence [63]. As it emphasizes the behavior linkage between parents and offspring, genetic recombination is not carried out in the evolutionary process. EP has now been applied to solve many numerical and combinatorial optimization problems.

In EP, offspring are created by mutating individual solutions in the current population according to a probability distribution. Then selection is made from the mutated and current solutions to form to a new generation. This selection is usually made through a stochastic tournament procedure.

Further, mutation in EP can be performed in different ways. In the standard EP, a parent solution is mutated by random numbers generated from a normal probability distribution. Adaptive scheme was developed in *R-meta-EP* [64] (an extension of the standard EP) to adapt both standard deviation and covariance matrix of the normally distributed mutations. Yao and Liu proposed another EP algorithm, called Fast Evolutionary Programming [65], by using Cauchy probability distribution in replacement of normal distribution to produce offspring in the larger neighborhood. Later, Levy probability distribution was adopted in [66] as a generalization of the Cauchy-based mutations to achieve higher variation and diversity of the search.

5.5 *Estimation of Distribution Algorithms*

Estimation of Distribution Algorithms (EDAs) [67] [68] differ from most other EAs in that evolution from one generation to the next one is not done by combination of previous solutions, but by estimating the probability distribution of the fittest individuals, and then sampling the generated model to generate new solutions. Thus, the generation of new solutions is done according to the probability distribution of best solutions, thereby avoiding the use of combination operators. However, estimating the probability distribution associated with the set of the individuals selected from the previous generation constitutes a hard work to perform.

Several EDA approaches have been developed, in function of the probabilistic models used and the methods employed to learn them. They could be divided into: *univariate EDAs*, which assume independency between the variables, like PBIL; *bivariate EDAs*, which take into account some pairwise interactions, like MIMIC; and *multivariate EDAs*, which consider more complex interactions, like Bayesian Optimization Algorithm, BOA [69]. In [70] and [68] there are more details. EDAs have been applied successfully to several types of problems [71], [72].

5.6 Particle Swarm Optimization

Particle swarm optimization (PSO) algorithms [73] [74] mimic the flocking behaviors of animals in their movement. Similar to GAs, PSO algorithms work with a population of particles, in which each particle contains a feasible solution to the problem and its velocity. The particles move around in the search space to improve their fitness (objective values) iteratively. The movement of each particle is determined in terms of both its best position in the history and also the best position known so far from all particles. In view of this, the speed of a particle at iteration k+1 is updated as:

$$v_{k+1} = w \cdot v_k + c_1 \cdot r_1 \cdot \left(P_{pb}^k - X_k \right) + \\ + c_2 \cdot r_2 \cdot \left(P_{gb}^K - X_K \right) \tag{8}$$

In Eq. (8) P_{pb} and P_{gb} denote respectively the best position of the particles and the best known position from all particles, w is the momentum, X_k is the position of the particle at iteration k, r_1 and r_2 are two randomly generated positive numbers, and c_1 and c_2 are the parameters used to balance the individual and social influences. Although it is a metaheuristic originally designed for continuous optimization, other discrete versions have been proposed [75].

The Comprehensive Learning PSO, CLPSO [76], was proposed as a variant of the standard PSO. Instead of using the best known position as in the original PSO, it uses the best within a group of P_{pb}. Its equation is as follows:

$$v_{k+1}^d = w \cdot v_k + r \cdot \left(P_{fbest(i,d)}^d - X_k^d \right) \tag{9}$$

where $P_{fbest(i,d)}$ defines which particle's P_{pb} the current particle should follow. This variation is very popular by its good results in many problems.

5.7. Differential Evolution

Differential evolution (DE) [77] [78] is a stochastic and meta-heuristic technique that has been developed for solving optimization problems with real parameters. It provides a powerful tool for searching for optimal solutions in high-dimensional spaces that are nonlinear, non-differentiable, non-continuous, and containing multiple local optima. DE is similar to GAs in the sense that both are evolutionary, population-based algorithms. But DE algorithms differ from GAs in the way evolutionary operators are manipulated to produce new child solutions. The main loop of DE algorithms is briefly explained in the following.

A DE algorithm maintains a population of real-valued parameter vectors and works iteratively. Each iteration starts with mutation, in which three distinct parameter vectors are randomly selected for every population member. The weighted differences between two parameter vectors are added to the third parameter vector to get the perturbed vector, as indicated in Eq. (10).

$$V_i = X_{r1} + F \cdot \left(X_{r2} - X_{r3} \right) \tag{10}$$

Then crossover is done to combine the population member and the perturbed vector to yield a new trial vector. Every parameter in the perturbed vector has a certain probability to enter the trial vector, following Eq. (11).

$$U_i^d = \begin{cases} v_i^d & if\ rand_d[0,1] \le R \\ X_i^d & otherwise \end{cases} \tag{11}$$

Finally the trial vector U_i replaces the old population member X_i if it has a lower objective value.

Because DE initially was too sensible to its parameters, several adaptive versions were proposed. SaDE [79] and JADE [80] are the most popular and successfully DEs. SaDE adapts automatically the parameters F and CR during the search, obtaining very good results in continuous optimization. JADE combines adaptation with an external memory of solutions to guide the search for results, showing an efficient algorithm for multimodal optimization.

DE attains increasing popularity in engineering applications due to its attractive features such as fewer running parameters to specify, ease in programming, high efficiency, as well as strong global search ability. A comprehensive review of various DE algorithms together with associated operators is given in [81].

5.8. *Artificial Bee Colony Algorithms*

Artificial Bee Colony algorithms (ABC) were recently proposed as a new group of metaheuristic search algorithms inspired by the behavior of a bee colony [82]. These algorithms use three concepts from bee colonies: F*ood sources, Employed foragers and unemployed foragers*. There are different food sources, and the value of each food source is different, represented by a quantity (They are the solutions to optimize). *Employed foragers* are associated with a particular food source which they are exploiting, thus they are responsible for the exploitation of current solutions/food source. *Unemployed foragers* are looking out for a food source to exploit. There are two types: *scouts*, searching randomly the environment for new food sources, and *onlookers*, that establish a food source by the information shared by employed foragers.

Although this type of algorithms is very recent, it is rather popular and presents an increasing activity of research in this category. A good and very actual review of ABC algorithms is given in [83].

5.9. *Memetic Algorithms*

Memetic algorithms (MAs) [84] [85], or Memetic Computing, are population-based metaheuristic search methods inspired by the principle of natural evolution and Dawkin's notion of memes capable of local adaptation. MAs are hybridization of metaheuristics and local search methods. According to the idea of Lamarckian learning, local search can be done to improve the found solutions, focusing the population algorithm on the exploration of the domain search. In real-world problems, local search can use information about the problem [86], however there are well-known local search algorithms that can be applied without any additional information [87].

As hybridization of evolutionary algorithms and local search, the aim of MAs is to exploit the best search regions gathered by global sampling with the population algorithm. Hence an important demand for MAs is the synergy between the exploration abilities of the population algorithm and the exploitation abilities of the local search [88].

For discrete or combinatorial optimization, MAs have proven to be the best alternative in problems like TSP or QAP [89]. In these problems, MAs tend to combine searches for exploring the entire decision space and searches which focus on portions of the decision space. Local search in MAs for discrete optimization performs an intensive exploitation of the search space using information about the problems, enhancing the performance. In [90] there is a survey about hybridization in combinatorial optimization.

For continuous MAs it is very important to decide to which and how long the local search would be applied. The local search method can be randomly selected [87] but the intensity is a difficult issue. The intensity can be applied in function of the solution. It can be made by levels [91] or applying the local search several times over the same solution, creating *LS Chaining* [92].

MAs are especially interesting in continuous optimization because they have shown to obtain very good results. During the last years, in the IEEE Conference on Evolutionary Computation, CEC, many special sessions have been organized. In these special sessions the organizers give the benchmark and the experimental conditions, to allow comparing the results of the different proposals. In these special sessions on continuous optimization, the majority of the best algorithms were MAs. In CEC'2013 [93] one winner was a multi-start algorithm, NBIPOPaCMA [94], but the other two were MAs: ICMAES-ILS [95] combining an ILS with IPOP-CMA-ES as global search algorithm and DRMA-LSCh-CMA [96], using local chaining and a new niching technique. In CEC'2014 [97], there are again two MAs among the three winner algorithms: GaAPADE [98], the first winner, a hybridization of an GA, an DE and an Evolutionary Strategy; L-SHADE [99], a hybridization of SHADE with LS and a reducing population (SHADE without LS got the fourth place in CEC'2013, clearly the LS improves the results). MVMO [100] was the only non-MA among the winners in CEC'2014.

In [101] there is a good review of MAs, for more details.

5.10. *Ant Colony Optimization*

Ant colony optimization (ACO) [102] [103] mimics the behavior of a colony of ants in searching for food. Prior to applying ACO, the optimization problem has to be transformed into the problem of path finding on a graph. Then a group of ants work collectively to find a shortest path on the graph by pheromone communication during path formation [104].

An ant builds its path incrementally. It starts from a randomly selected vertex and then chooses an edge to go to the next vertex. The choice of an edge is stochastic yet its probability is decided by the pheromone values and heuristic information associated with the edge. The most well-known rule for determining the selection probability for edge c_{ij} is given in Eq. (12)

$$P(c_{ij}) = \frac{\tau_{ij}^{\alpha} \eta_{ij}^{\beta}}{\sum \tau_{il}^{\alpha} \eta_{il}^{\beta}} \tag{12}$$

where S_f denotes the set of feasible edges immediately after the current partial path, τ_{ij} and η_{ij} are the pheromone and heuristic values respectively associated with edge c_{ij}, α and β are the parameters controlling the relative importance of pheromone versus heuristic information.

Further, when the ants completed their paths, the quality of their solutions is used to update the pheromone values of the edges. These updated values are then utilized by the ants in the next iteration to build new paths. This procedure continues until the maximum iteration number is reached or all ants tend to produce similar paths (i.e. the algorithm has converged).

In [106] there is an interesting review of ACO algorithms tackling different engineering domains.

5.11. *Scatter Search*

Scatter Search [107] [108] is an optimization algorithm, especially applied in combinatorial optimization, but it can be applied in continuous optimization as well. It operates on a set of solutions, the *reference set*, that are combined to create new ones (using the subset of the best solutions as parents to guide the search). It also uses several techniques to enforce the exploration and a local

search method is often employed as a mechanism for improvement.

Path relinking is a very important concept in scatter search. The idea is to consider the path between solutions in terms of moves that separate both solutions. Thus, based on two solutions, a number of new intermediate solutions can be generated following the path that connects them: starting from an *initiating* solution, several moves are performed to reduce the distance to a *guiding* solution. The role of *initiating* and *guiding* solutions are interchangeable. This technique can produce better solutions, and several other combinatorial algorithms, like GRASP [109] or Tabu Search [110] can also be incorporated here to further improve the results.

6. Current Important Issues

Various (basic) metaheuristic techniques have been addressed in the preceding sections. They target at the optimization problems with only one objective function and only one global optimum. However, more challenging situations will be encountered in real applications. One is due to the multi-modal nature of many practical problems that contain several global and local optima to identify. The other is caused by multi-objective optimization tasks that entail discovering a set of trade-off (Pareto-optimal) solutions. Moreover, high computational cost is often needed when applying metaheuristics, thus it is would be very beneficial to have parallel version of the metaheuristic algorithms to be able to run in in multiple computers to locate good solutions as fast as possible. Multi-modal optimization, multi-objective optimization, as well as parallel metaheuristics represent three important issues for the research community, which will be carefully discussed in the remaining of this section.

6.1 *Handling Multi-Modal Optimization Problems*

The objective in multi-modal optimization problems is to find a number of global or local optima. One strategy to achieve this is to restart the optimization process many times, intending to obtain a new optimum at each restart. Unfortunately, one could possibly obtain the same result from a previous search. The other way is to locate a set of different optima all together from a single optimization process, which appears more attractive and useful and will be addressed in this paper.

Population-based optimization techniques maintain and deal with a set of possible solutions simultaneously during the search process. Nevertheless they were originally designed to locate a single global optimum rather than multiple optima. This limitation can be overcome by some numerical techniques, commonly known as niching methods. A niching method modifies the behavior of a traditional (population-based) algorithm to maintain solutions in different areas of the domain search to avoid convergence to only one area. That imply a greater diversity of the population for support of discovery of multiple optima from a single run of the algorithm.

6.1.1 *Niching Methods*

The concept of niching is inspired from the nature in which organisms can only survive in the regions to which they are specially adapted. In optimization algorithms, niching methods are used to maintain subpopulations in a population such that convergence to multiple optimal solutions is possible. The commonly used niching methods include crowding, fitness sharing, clearing, speciation and clustering, which will be briefly introduced below.

De Jong [111] proposed the original crowding method, in which an offspring competes with the most similar individual from randomly sampled individuals from the population. This can be used to select similar solution for replacement, like *deterministic crowding* [112], or to select randomly several solutions to compare and choose one of them, like the restricted competition selection method, RTS [113]. The number of solutions compare is called *crowding factor*.

The fitness sharing method was introduced by Holland [114] and Goldberg & Richardson [115]. Instead of using directly the fitness f of an individual, it uses the modified version f_s defined as follows:

$$f_S(i) = \frac{f(i)}{\sum_{j=1\cdots N} Sim(d_{ij})} \tag{13}$$

where $f(i)$ and $f_s(i)$ are the original and modified fitness functions respectively and *Sim* is a similarity function defined as

$$\mathrm{Sim}(d_{ij}) = \begin{cases} 1 - \left(\dfrac{d_{ij}}{\sigma_{share}} \right)^{\alpha} & if\ d_{ij} < \sigma_{share} \\ 0 & otherwise \end{cases} \quad (14)$$

where d_{ij} is the distance between individuals i and j, N is the population size, α is the sharing level and σ_{share} is the *sharing radius* which specifies the threshold of distance for two individuals to have nonzero similarity. It is very sensitive to the *sharing radius*.

Clearing method [116] attempts to select only the best individual while discarding the inferior ones in each niche. This method removes the solutions that are too close (using a minimum distance among solution, called *niching radius*) to a better solution. This leads to higher diversity of the population since a number of similar solutions are discarded in a clearing procedure, but it has the disadvantage of been very sensitive to the *niching radius*.

Speciation has been commonly used in multimodal optimization tasks to maintain diversity and stable niches over generations [117, 118]. It is realized by dividing the population into different species according to similarity among its individuals. Each species is formed around the *species seed* that has the highest fitness value among all the species members. All individuals that fall within the species radius from the species seed are considered to belong to the same species. The main disadvantage of the speciation method lies in the difficulty of selecting a proper value for the radius parameter.

Single Linkage Hierarchical Clustering is a technique to help building subswarms for particle swarm optimizer in a dynamic environment [119]. Compared with *k-means* clustering [120], it can adaptively adjust the number of subswarms needed to automatically identify the promising search region for each subswarm.

6.1.2. *Niching Optimization Algorithms*

Niching methods have been widely applied with the population-based optimization techniques to ensure maintaining solutions in different areas of the domain search, creating a high diversity of the population to support searching multiple optimal solutions simultaneously. Next we discuss some of the works done in this direction.

Initially, the niching techniques were used with genetic algorithms creating sharing genetic algorithm and clearing-based genetic algorithm [121]. Unfortunately, both of them have a strong dependency of the niche ratio. Lin and Wu [122] presented the use of a clustering technique to identify the radius of each niche before fitness sharing is performed. In [123], a dynamic fitness sharing is proposed that estimates dynamically the niche ratio considering the distance between solutions. There have been proposed also adaptive hierarchical niching technique [124] capable of obtaining good results [125]. Another option that gives extraordinary good results is to use a hybridization of different populations, each one with a different niching technique (or parameter) [126].

Perez, Posada and Herrera [127] made an empirical study to compare different niching genetic algorithms in terms of efficacy, multi-solution based efficacy (the capability to find multiple optima) and diversity in the final set of solutions. The results of this study demonstrated that a niching GA having a certain type of replacement process performed much better than its counterpart without including that replacement.

Although initially these techniques were used with GAs, several niching DE algorithms have been proposed to enhance the diversity among population members in tackling multi-modal problems. Thomsen [128] studied the integration of the fitness sharing scheme and the crowding mechanism with DE, revealing that the crowding DE was superior to sharing DE when the crowding factor was equal to the population size. The crowding concept is also adopted in the crowding-based DE algorithm proposed in [129], where an offspring replaces the most similar individual from the population if the fitness value of the offspring is better. Besides, a modified fitness sharing DE algorithm was introduced in [130] for comparison among various DE variants.

Species-based algorithms utilize information from different niches of the old population to create a new generation with good diversity, different species to acquire localized convergences towards multiple global optima. The species are determined using species radius. This technique have been used with genetic algorithm [131] DE [132], or SPSO [133]. The problem with these algorithms is that it is not a trivial task for users to specify proper values of this parameter in real application.

The introduction of DE gives the possibility of developing new niching techniques. Qu, Suganthan and Liang [130] proposed a neighborhood-based mutation strategy for niching DE algorithms. Following this strategy, both the base individual and individuals for generating difference vectors are selected from a local neighborhood. This encourages members of the population to converge towards the optimal points in their local niches. Beneficially, the neighborhood size for mutation is easy to specify.

The use of an external archive can be used to maintain all the solutions found so far and it also encourages continuous search in unexplored areas via re-initialization for offspring already in the archive. JADE [80], using an external memory, in conjunction with the adaptation of its parameter, obtains very good results.

In 2013, an algorithm was proposed using an alternative method for mutation, DE/nrand/1 [134], that was one of the winners in the niching competition in CEC'2013 [135]. It uses a scaled difference of two random population members to mutate the nearest neighbor individual, in combination with an external archive to guide the search, and a restart mechanism. Another winner, DRMA-LS-CMA [96] proposed the idea of creating the niches dividing the domain space into hypercubes, to avoid the computational cost of the Euclidean distance.

In PSO there have been proposed also several niching algorithms. The main idea of niching PSO algorithms, like NichePSO [136], is to update the position of a particle with the influence from the best particle in its neighborhood rather than the best one from the whole swarm. Hence, Eq (8) for particle speed updating in the original PSO is adapted to the following form:

$$V_{K+1} = W \cdot V_k + c_1 \cdot r_1 \cdot \left(P_{pb}^K - X_k \right) + \\ + c_2 \cdot r_2 \left(lbest(K) - X_k \right) \quad (15)$$

where *lbest*(*k*) denotes the best position in the neighborhood of the current particle at iteration *k*.

The niched ant colony optimization algorithm was addressed in [137], with the division of the ant colony into a set of distributed niches. Each niche-colony had its own pheromone matrix in evolving towards an optimal solution. The multiple pheromone matrices connected with different niches led to higher diversity among individual solutions in comparison with standard ant colony algorithms.

CMA-ES niching algorithms [138, 139] were proposed with the intention to employ standard CMA-ES locally for different portions of the population with each subpopulation carrying and updating its own covariance matrix and step size. In [138] the niches were built using individual niche radius that was adapted for each individual along with its adaptive strategy parameters. Following this idea, NEA2 [139] adopted the nearest-better clustering algorithm as a radius-free approach to identifying different niches of the search space to run CMA-ES locally. This algorithm was the winner of the niching competition on CEC'2013 [135].

Another important trend is the hybridization combining DE and PSO. Initially one of them was used as a LS method to the other or as a perturbation method to avoid premature convergence. But, nowadays there are more models using other cooperation techniques such as co-evolution. Readers can find a review of this type of hybridization in [140].

In [141] there is an interesting review of evolutionary algorithms for multimodal optimization.

6.2. *Handling Multi-Objective Optimization Tasks*

In complex engineering problems there is often more than one objective to consider for deciding optimal solutions. Generally, a multi-objective optimization problem (MOP) can be stated as

$$Minimize\ F(X) = \left(f_1(X), f_2(X), \cdots, f_m(X) \right)$$
$$\text{Subject to } X = \left(x_1, x_2, \cdots, x_n \right) \in \Omega$$

where X=(x_1, x_2, …, x_n) is the vector of design variables, Ω denotes the region of feasible solutions in the decision (variable) space, and F(X) is an objective vector containing a set of real-valued objective functions f_1, f_2, …, f_m.

Let be two feasible $X_1, X_2 \in \Omega$ solutions, X_1 is said to dominate X_2 if and only if $f_i(X_1) \leq f_i(X_2)$ for any

$i \in \{1,2,\cdots,m\}$ and $f_j(X_1) < f_j(X_2)$ for at least one index $j \in \{1,2,\cdots,m\}$. A solution $X^* \in \Omega$ is Pareto-optimal if there is no other $X \in \Omega$ such that X dominates X^*. The objective vector $F(X^*)$ of a Pareto-optimal solution X^* is called a Pareto optimal objective vector. The set of all Pareto-optimal solutions is called the Pareto set, and the set of all Pareto optimal objective vectors, as shown in Fig. 1, is termed as the Pareto front [142].

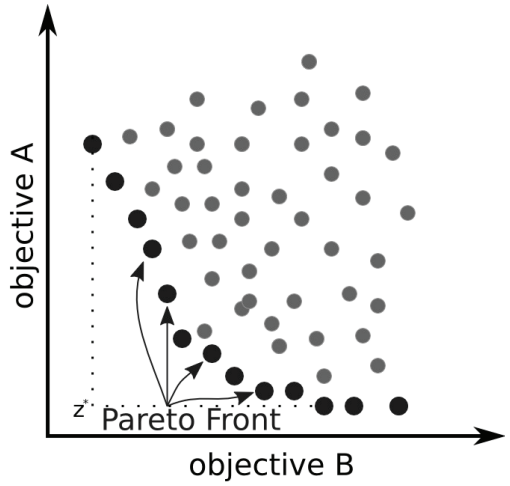


Fig. 1. Pareto front with two objectives

MOPs in real applications can have many or an infinite number of Pareto optimal solutions. It is important to present a manageable number of them to a decision maker for the final choice. The selected Pareto optimal solutions should yield evenly distributed samples in the objective space, thus offering a good approximation of the Pareto front.

As for discovery of Pareto optima, there are basically three categories of methods: one is based on the aggregation of multiple objectives into a scalar function; other is based on decomposition to handle solutions of the subproblems at the same time; and the third uses dominance relation among solutions. They will be outlined in subsections 6.2.1, 6.2.2 and 6.2.3 respectively.

6.2.1. *Optimization Based on a Scalar Function*

In the weighted sum approach [142], every objective is assigned with a weight in terms of its importance and the overall function is built by a weighted average of the objectives. Let $W = (w_1, \cdots, w_m)$ be a weight vector with $w_i \geq 0 \; \forall \, i = 1, \ldots, m$ and $\sum_{i=1}^{m} w_i = 1$, the combined function to optimize is formulated as follows

$$g_1\langle X|W\rangle = \sum_{i=1}^{m} w_i f_i(X) \tag{16}$$

Obviously, any solution that minimizes the overall function in Eq. (16) is a Pareto-optimal solution to the original MOP. This means that a set of Pareto-optimal solutions (for the MOP) can be acquired by optimizing the function in Eq. (16) under different weight vectors. However, when the Pareto front is not convex, not all Pareto optimal solutions can be found by using this method.

In the Tchebycheff approach [142], the overall objective function is formulated as

$$g_2\langle X|W,Z^*\rangle = max_i\left\{w_i\left(f_i(X) - z_i^*\right)\right\} \tag{17}$$

where $Z^* = (z_1^*, \cdots, z_m^*)$ is a reference point, which is characterized by the following expression $z_i^* = min\langle f_i(X) | X \in \Omega \rangle \; \forall \, i \in \{1, \cdots, m\}$.

Minimization of the function in Eq. (17) with a specified weight vector gives rise to a Pareto optimum for the original MOP. Different values of the weights can lead to different Pareto optima. Further, every Pareto optimum is obtainable by seeking the best solution for Eq. (17) with a set of properly defined weights. One weakness with this approach is that its aggregation function is not smooth for a continuous MOP.

As stated above, good samples of the Pareto set can be created by processing the aggregated overall function with diverse weight vectors. A particular set of weights adopted in the overall function corresponds to a sub-problem of the original MOP. Traditionally, such sub-problems for the MOP are solved in a sequential procedure. However this sequential manner is not so attractive in practice due to its low efficiency.

The population-based optimization approaches have the potential to be adapted for dealing with multiple sub-

problems of the MOP at the same time. The multi-objective genetic local search [143], [144] was developed to simultaneously optimize all combinations of multiple objectives via either the weighted sum approach or the Tchebycheff approach. It generates a random weight vector at each iteration to construct the aggregated function for fitness evaluation.

6.2.2. *Optimization Based on Decomposition*

Zhang and Li [145] proposed an evolutionary algorithm based on decomposition (MOEA/D) to solve the subproblems of the MOP at the same time. The population consists of the best solutions found so far for each subproblem. In the evolutionary process, the solution for each subproblem is improved by utilizing information only from the neighboring subproblems, which greatly reduces computational complexity of the algorithm. This work also obtained the Outstanding Paper Award of IEEE Transactions on Evolutionary Computation in 2010.

Since decomposition may result in tasks with different computational difficulties, different amounts of computation could be carried out on different subproblems, as it was done in MOEA/D-DRA [146]. That algorithm was the winner for unconstrained problems in the Competition for Multi-Objective Evolutionary Algorithms in CEC'2009 [147].

In recent years, other extensions of the MOEA/D algorithm have been proposed, such as hybridization of MOEA/D with DE [148] and with ACO [149] respectively. Besides, parallel implementation of the MOEA/D algorithm was developed in [150].

6.2.3. *Optimization Based on Dominance Relation*

The third category of approaches to MOPs attempt to search for a diverse set of Pareto optimal solutions by directly considering multiple objectives rather than aggregated evaluation functions. In principle, population-based optimization techniques can serve well this purpose since they do exploration in parallel and can handle multiple objectives at the same time. For instance, multi-objective genetic algorithms (see examples in [151], [152]) were developed and demonstrated as capable of guiding the search in light of multiple objective values within a single running of the evolutionary process.

However, conventional population-based techniques were originally designed for optimizing a single fitness function. They have to be modified to exploit solution dominance and other relevant information in the selection operator. As one of the earliest efforts in this direction, Zitzler and Thiele [153] proposed the Strength Pareto Evolutionary Algorithm (SPEA), in which the fitness of an individual in the population is assigned according to the archive members that dominate it. Later, this algorithm was further developed into an enhanced version called SPEA-II [154]. The SPEA-II algorithm improves the original one in the following aspects. First, evaluation of an individual considers not only the individuals dominating it but also those dominated by it. Second, density value is incorporated into the fitness assessment to enable more precise guidance of search. Third, a better archive truncation method is used to preserve boundary solutions.

In the non-dominated sorting genetic algorithm (NSGA) proposed by Srivinas and Deb [152], population members are first classified according to dominance relation into different levels of rank, and individuals in the same rank are assigned with an equal fitness value. Then all such fitness values are modified using a fitness sharing method to encourage an even selection of parents for creation of offspring. The Fast Non-dominated Sorting Genetic Algorithm (NSGA-II) [155] is based on NSGA but it has two major improvements over the original one. First, it employs a fast non-dominated sorting algorithm to speed up the classification of population members into different ranks. Second, it introduces the crowding-distance metric to enable the comparison and selection of individuals at the same rank to enter the next generation.
In [156] and [157] a new extension was proposed, NSGA-III, focused on many objectives optimization.
NSGA-III differs from previous one the maintenance of diversity among individuals that aided by supplying and adaptively updating a number of well spread reference points. In this way it is more adequate for optimization with more objectives than NSGA-II. NSGA-III has been successfully applied for box constraints and general constraints problems.
The dominance relation is also useful for PSO algorithms to select global and local best particles in face of multiple objectives. Different selection strategies have been proposed for this purpose. In [158] the tournament niche method was used to decide the global best particle, and the local best particle was identified

according to Pareto-dominance. The other interesting strategy is to stochastically choose the global best particle from the non-dominated solutions using density-based probabilities [159].

The Pareto Differential Evolution Approach was proposed in [160], which calculates the non-dominance rank and crowding distance for the combined population consisting of both individuals from the current population and newly created offspring. Then the best individuals are selected for entering the next generation in terms of the non-dominance rank of individuals as well as the diversity metric (crowding distance) for solutions on an identical rank. Xue et al. [161] did a similar work in using the non-dominance rank and crowding distance to distinguish individuals in the combined population. Further, they converted non-dominance and diversity information into fitness scores of solutions to guide the choice of the most competent individuals into the next generation.

Robic and Filipic [162] developed a DE for MOP, in which the treatment of a new solution depends on the result of comparison with its parent solution. The new solution immediately replaces its parent if it dominates. Otherwise, if the parent solution dominates, the new solution is discarded. In case when both solutions are not dominated, the new solution is added into the current population without replacement of its parent. After processing all new solutions, the whole population is truncated in terms of the non-dominance rank and crowding distance to select a fixed number (population size) of individuals to survive in the next generation. Compared with the work in [160], the main merit of this algorithm lies in the immediate replacement of an inferior parent solution by a new stronger one such that the new stronger solution can take part in producing other new solutions. However, discarding dominated solutions in the algorithm could cause loss of information (good solutions) when evolving a set of solutions from one generation to another. To overcome this weakness, Ali et al. [163] proposed an improvement scheme that maintains a secondary population to store those candidate solutions that are temporarily excluded from the current population. Every newly created solution enters the secondary population if it does not dominate the parent solution. Otherwise, when the new solution dominates the parent solution, the new solution is added into the current population while its parent is moved to the secondary population. Finally, the individuals of the new generation are selected from the mixture of the current and secondary populations following the same selection criterion as that used in [160].

Another Pareto-frontier differential evolution algorithm was presented in [164], which suggests selecting the parents for mating from the set of non-dominated solutions. The Pareto-adaptive ϵ -dominance was used in [165] for updating the external archive and saving extreme solutions for a multiobjective DE algorithm. The parents in this DE algorithm are selected alternatively by a random scheme and an elitist scheme.

6.3. *Parallel Metaheuristics*

The majority of implementations of metaheuristics are run sequentially, even when parallel-based algorithms search in parallel. However, the introduction of parallelism can reduce the search time and improve the quality of the solutions, specially nowadays when every computer has several processors. There are several specific reviews about parallel metaheuristics [166], [167], [168], and about the parallel version of certain algorithms [169], [170]. In this subsection we are going to make a brief survey of them, which readers can consult for more details.

For trajectory and multi-trajectory metaheuristics, there are mainly three different ways of incorporating parallelism in the literature: parallel evaluations of neighbors (or new solutions), in which the behavior of the algorithm is the same; parallel evaluation of a single solution, especially interesting in very time-consuming evaluations; and parallel multi-start models in which several trajectory-based methods are running at the same time [171]. The last one is the only one among them that can offer different results than sequential implementation of the algorithm. The searchers can be homogeneous, or heterogeneous, share information or not, there are many possibilities.

For population-based algorithms, there are two strategies, to parallel the operations over the same population (evaluation, Euclidean distance calculation, etc.) or split it into several subpopulations, which is called *island model* [172]. The island model is able to tackle niching problems, and usually there is a migration policy to share information for improving the search, such that results of one subpopulation (as the best current solution) could guide the other subpopulations. An extreme model is called cellular method [167], designed for high-parallelism in which one solution can only com-

municate with its neighbors. Sometimes these models can be combined.

6.3.1. *Island Model and Cellular Evolutionary Algorithms*

The island model is one of the most used parallel models. In them, the population or swarm is divided in several subpopulations/subswarms, as shown in Fig. 2. Originally it was proposed for Genetic Algorithms [172], this model is now also used in DE [173][174] and parallel PSO [175].

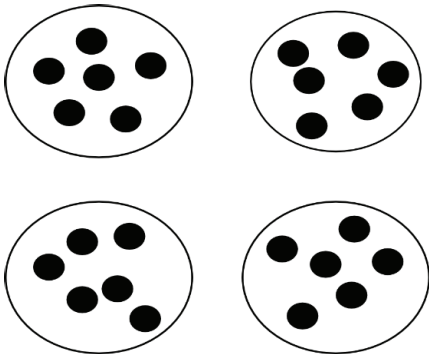


Fig. 2. An island with subpopulations/subswarms

In this model, each subpopulation evolves in parallel with the others, thus the algorithm can be run in different processes, each of them working on a different subpopulation. The best improvement could be obtained with a different processor for each subpopulation, but it can also be used with less processors than subpopulations.

These subpopulations shares information for guide better the search. This sharing of information implies many decisions: the topology (the connections between islands) [176], the frequency and which solutions should be exchanged [177], and how the new solutions should be integrated into the population.

The above model produces distributed evolutionary algorithms, dEAs, or *coarse-grain,* in which each solution can communicate with all other individuals in the same subpopulation. There are also fine-grain algorithms or *cellular evolutionary algorithms*, cEAs, in which individuals can only interact with their neighbors in the reproductive cycle where the variation operators are applied. This reproductive cycle is executed inside the neighborhood of each individual and consists in selecting parents among its neighbors, applying some operators to them, and replacing the individual by the created offspring following a given criterion (usually when it is worse than the new one). These algorithms based on the Cellular Model (as shown in Figure 3) can be parallelized in more processors than the original *island model.*

6.3.2 *Technology for Parallel Metaheuristics*

The design of a parallel algorithm cannot be completely separated from the hardware architecture in which the algorithm is going to be implemented. Flynn [178] proposed the division in several categories, in which the more relevant are: *multiple instruction-multiple data, MIMD*, in which several autonomous processors are used, a multi-core system or a distributed system; and single instruction-multiple data, *SIMD*, in which the same instructions can be run at the same time over several data.

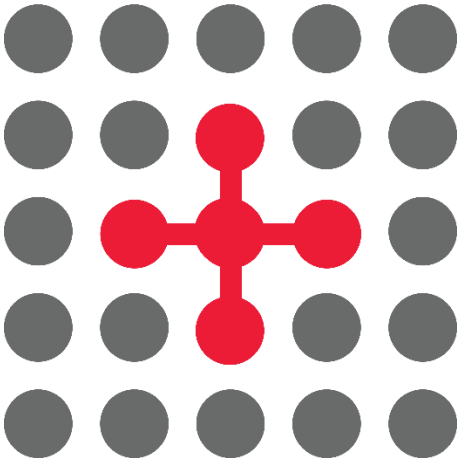


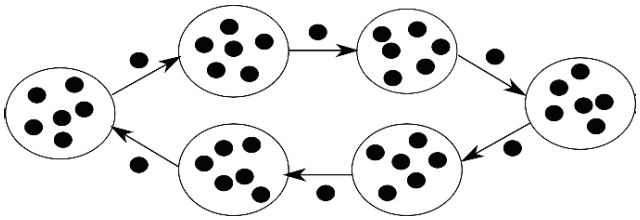
Fig. 3. Cellular Model where each solution is only connected with its neighbors.

The MIMD model is implemented by a multi-processor architecture. One popular option is the use of multi-core systems because they allow using shared memory for the data, a very efficient and simple way of communication. Another option is using autonomous computers, in which there is no shared memory and the information has to be transmitted by the network. These two systems can be used together, which is very usual nowadays.

The parallel programming of multi-processor system is the most classic model. One can use solution ad-hoc using TCP sockets (or any messaging library) to communicate different systems, and the utilities from the operative system (threads) for multi-core communications.

However, it could be difficult to reuse the solution. Another better option is to use a higher library for developing parallel computing applications that allow one to concentrate more on the programming, and less into the communication. The most popular standard for scientific computation in distributed system is the MPI standard with several implementations [179] and OpenMP [180] for shared memory systems. Also, one can use some of the frameworks described in Section 7.

The SIMD model is implemented in systems in which



the same instruction can be broadcast to all processors, applying the same operation to different data. Requiring fewer instructions to process a given mass of data, SIMD operations are very efficient in processing data, which is very important in many mathematical operations.

The most popular way of programming in SIMD is by Graphics Processing Units (GPUs). These elements included in the majority of current graphical cards were designed for improving the complex calculations required for image processing for modern video-games. However, they can be used for general purpose algorithms. General purpose computation on GPUs (GPGPU) [181] allows algorithms to perform parallel computations over different data using the general purpose computing capabilities of modern GPUs. Recently, several parallel EAs for optimization using GPUs have been published, such as PSOs [182] or DEs [183][184].

The GPGPU computation discipline has been a very active research topic in the last years, especially since popular computing frameworks like CUDA or OpenCL were introduced. These platforms have allowed for using the great computing capabilities of modern GPUs for general purpose problems by using extensions of high level programming languages. CUDA is the most popular, it is the platform provided by NVIDIA which allows for developing applications on popular NVIDIA GPUs using a subset of C/C++ (Fortran and other programming languages are also supported) with some extensions that provide access to the GPU.

6.2.3 *New models of Parallelism*

In the beginning, the well-known structure *master-slave* was used, in which one central processor carried out the task with the help of a group of slave processors that run several actions in parallel. In this model the number of slaves could be increased, but the master is the bottleneck. Another alternative are distributed systems, in which all computers run the algorithm with a small population with frequent exchange of individuals between them, following the island model. In this model, all computers have the same responsibility. However, the communication between them requires a topology. This topology limits the growth of the model, creating a static communication structure that should be adequate for the hardware resource available for an efficient run (i.e. a processor for each subpopulation).

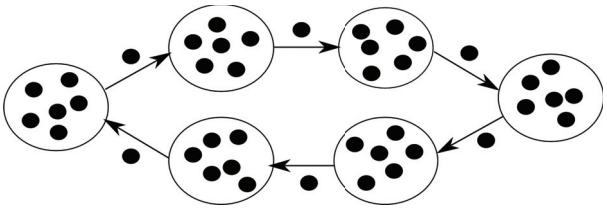


Fig. 4. Distributed systems.

In recent years, inspired by the new networks possibilities, new models have arisen. One of them is inspired from the peer-to-peer platforms, P2P. In a P2P system there is no centralized component, each component shares information only with its neighbors, and the number of components is not defined *a-priori*. But there are unstructured peer-to-peer networks in which the number of nodes can change dynamically [185]. This model and technology have obtained good results in very scalable systems. This technique can be used in conjunction with a parallel EA, to create peer-to-peer EAs [186]. In these models, because there is a static neighborhood, selection is locally made using the current neighborhood. P2P EAs are a promising approach to hard optimization problems with a greater scalability [187].

Another new model is Cloud Computing [188]. In this model there are available remote resources available on demands, allowing using/paying the resources on a short-term basis as needed (for example, processor by hours and storage by the day) as releasing when they are no longer useful. In a parallel algorithm, it is very interesting, because we can run a distributed algorithm

without requiring a specific infrastructure, and it is very scalable [189]. Nowadays, there is a start with the emergence of EAs that use cloud computing for solving very hard problems [190], [191].

7. Software Support Tools

Recently a number of software systems have been developed that provide customizable tools for implementing optimization techniques to solve various practical problems. They bring benefits to both expert and non-expert users in project development, saving time and cost. They can also support the evaluation and comparison of different metaheuristic optimization methods to tackle a specific problem at hand.

Parejo et al. [192] conducted a comparative study of 10 selected metaheuristic optimization frameworks as listed in Table 1. The adopted criteria for comparison cover six areas of interest, including: C1) Metaheuristic techniques; C2) Adaptation to the problem and its structure; C3) Advanced characteristics; C4) General optimization process support; C5) Design, implementation and licensing; C6) Document and support. Every area is further divided into a number of features and each feature is assigned with a weight reflecting its importance. These weights are used in evaluations to calculate the scores of a framework in different aspects.

The study in [192] indicates that ECJ achieves the best overall performance among the 10 compared frameworks. However, ECJ still has to be improved in areas C1 and C5 as its scores are below the average. ParadisEO, with excellent performance in areas C1 and C3, is assessed as the second best in terms of all the evaluation criteria as a whole, yet it is scored below the average in C4 area. According to criteria C1 alone, FOM and ParadisEO are the best candidates since they offer the most support to the realization of various metaheuristics.

Table 1: The 10 metaheuristic optimization frameworks (MOFs)

MOFs	Prog. Lang.	Platforms
EasyLocal [193]	C++	Unix
ECJ [194]	Java	All
ParadisEO [195]	C++	All
EvA2 [196]	Java	All
FOM [197]	Java	All
HeuristicLab [198]	C#	Windows
JCLEC [199]	Java	All
MALLBA [200]	C++	Unix
Optimization Algorithm Toolkit [201]	Java	All
Opt4j [202]	Java	All

MOEA (http://www.moeaframework.org) is another well known framework that contains free and open source Java library. It is mainly designed to support developing and experimenting with multi-objective evolutionary algorithms and other general-purpose optimization techniques. Apart from including a set of base algorithms such as NSGA-II and MOEA/D, MOEA also contains a Service Provider Interface that enables new algorithms, problems and operators to be integrated into the framework.

8. Future Trends

Here we would like to point out that, owing to increasingly sophisticated application environments, the following issues are becoming highly important and they represent the new trends of research and development of metaheuristics optimization methods and systems.

8.1. *Large Scale Optimization Problems*

Large scale optimization problems appear very frequently in modern industrial scenarios, where the number of decision variables (or design parameters) tends to be extremely high (from several hundreds to thousands). The performance of many metaheuristic algorithms severely deteriorates as the size of the search space grows exponentially with the increasing number of variables. It is paramount to investigate new, more powerful methods and algorithms to tackle high problem dimensionality, to better explore the huge search space with only limited computational budget.

Cooperative co-evolutionary algorithms attains much interest to cope with complex optimization problems with many variables [203, 204]. They apply the divide-and-conquer approach to decompose a large scale problem into a set of low dimensional ones. Variable interaction is an important factor to consider for finding an appropriate decomposition for a particular problem. Recently, the differential grouping method [205] has been proposed for automatic identification of non-separable and separable variables for a co-evolutionary algorithm.

In the last years, several special issues [206] and special sessions [207] [208] were organized with the proposal of specific benchmarks for competitions on large scale optimization. The winners in these competitions were hybrid algorithms [209] [210] and cooperative algorithms [211]. The current best algorithm seems to be MOS [210], which combines DE with a specific LS method for large scale optimization and a hybridization mechanism is employed to dynamically decide the algorithm to apply based on a quality measure.

8.2. *Expensive Optimization Problems*

Expensive optimization problems are encountered in many engineering applications where the evaluation of a solution/design is often implemented through a time consuming procedure such as simulation or *finite element method*. Costly fitness evaluation would cause a prohibitively high computational cost for an optimization algorithm. A smart way to reduce the overall search time is to use computationally efficient models to partly replace the original fitness function during the optimization process. Such models are termed as approximate fitness models [212], surrogates or surrogate models [213]. Surrogate assisted optimization is becoming a hot and significant topic for improving computational efficiency of metaheuristic algorithms.

First of all, surrogates of good quality must be constructed. They have to approximate the original fitness function with reasonable accuracy to prevent the optimization algorithm from being misled into a false optimum. Building an ensemble of homogeneous or heterogeneous surrogates is useful to increase the accuracy in fitness prediction. Another method for accuracy improvement is resorting to dimension reduction [214, 215], i.e., to build a surrogate in a new, transformed space of lower dimensionality. Nevertheless, the effect of error from surrogates is not always negative. Sometimes such error can be utilized beneficially for smoothing the rugged fitness landscape and thereby accelerating the search process [216]. Therefore, mitigating model error and tolerating prediction uncertainty turn out to be two different aspects to take into account in building a set of surrogate models.

The other key issue concerns the surrogate management strategy for when and where to apply the surrogate models in replacement of the original fitness evaluations. One intuitive way is to perform original fitness evaluations on those trial solutions that are representative (according to cluster analysis) or potentially strong in fitness assessment. On the other hand, surrogate models are not good candidates for fitness evaluations when the models have a high degree of uncertainty in approximating the true fitness values. In some works [215, 216], surrogates were used only for search within a local area in an evolutionary algorithm. An important question for future research is how we could develop a systematic methodology for optimal usage of the surrogate models, such that the number of original fitness evaluations could be reduced as much as possible maintaining good results.

8.3. *Automatic Tuning and Self-Adaptation of Algorithmic Parameters*

Although metaheuristics are proved powerful problem solvers in wide practical applications, their performance heavily depends on the setting of their parameters. An improper assignment of parameters would lead to poor results when applied to a particular problem. Traditionally, adjustment of algorithmic parameters is done manually by trial and error, which is a tedious and very time consuming task.

Automatic parameter tuning refers to automatically finding a good set of parameters of the algorithm before its execution [217]. It can be considered as a complex optimization task. The search methods specifically designed for automatic parameter tuning include: Iterated F-Race [218], Iterated Racing Procedure [219] (as an extension of Iterated F-Race), Sequential Parameter Optimization [220], ParamILS [221], and Sequential Model-Based Optimization [222]. Automatic tuning enables thorough exploration of the algorithm design space that results in better parameter settings than those from manual tuning. Fine tuning parameters by computers can also provide an unbiased starting point for fair comparison of performance of different optimization algorithms.

Self-adaptation of parameters is also called parameter control [217]. It aims to adaptively change the values of the parameters in function of the results obtained during the execution of the algorithm. Usually adaptive behavior of parameters can make substantial improvement of the performance of an algorithm in

contrast to using constant parameter values. One example for this was demonstrated in the adaptive DE algorithm, SaDE [79]. More recently, a self-adaptive evolutionary optimization method, SaEvO [223], was proposed. The main part in SaEvO is a memetic algorithm (using DE and VPN) designed for solving the optimization problem, and an artificial immune system is integrated as the second part to adapt the parameters of the global search (DE) and local search (VPN) respectively during the optimization process.

Further, from the user perspective, it would be ideal to have parameter-free algorithm and not having that tune the different parameter to each problem. Many works are being done on self-adaptation of parameters within metaheuristic algorithms to ultimately reach this goal. The more an algorithm can adapt its parameters, the closer it meets the parameter-free requirement. The other way to advance towards parameter-free algorithms is to develop new rules, operators and mechanisms to be used in the algorithms such that the original parameters play no role, see examples in [225] and [224]. But the research in this topic has just begun with merely initial results.

8.4. *Synergy with Cloud Computing*

Cloud computing is an emerging computing infrastructure that provides flexible and on-demand access to a large pool of computational resources [188], [226]. It makes possible to easily use remote resources and thereby overcoming the limits by the local infrastructure. Costly computation and data intensive tasks can be conducted in a clouding environment to achieve shorter computing time while not incurring extra cost for hardware.

The rise of cloud computing produces significant impact on metaheuristics. It creates an opportunity to explore new implementations of metaheuristic algorithms by using remote and virtual computing resources. Indeed, parallel metaheuristics receive growing interests to solve complex and large scale optimization problems. With the availability of the cloud infrastructure, the computing task can be distributed to many virtual resources outside the local system, leading to great acceleration of the optimization process. However, the marriage of cloud computing with metaheuristics is just at the infant stage. How to manage the usage of cloud resources in terms of job scale and properties would be an important issue for research and application of cloud-based metaheuristics in complex optimization scenarios.

9. Relevant discussion topics

In this section, we are going to introduce several topics that often appear in the literature, because they are important for the success of metaheuristic or because they have been the object of discussion in recent years. The next list of items will be discussed in along the section:

1. The importance of a good trade-off between exploration/exploitation.
2. How memorizing additional information can improve the search.
3. Whether it is important that algorithms are nature-based or not.
4. The discussion between designing metaheuristics for one specific problem and designing them for a benchmark of problems.
5. The influence of the *Non-free luch theor*y.
6. New hybrid metaheuristics.

For reasons of space the discussion is brief, but we are cognizant that these issues require a thorough analysis due to their importance. The references provided allow the reader to have a deeper insight into these topics.

It is well-known that part of the success of a search algorithm is its trade-off between exploration of the search domain and the exploitation of the information obtained by the generated solutions [227], [228]. To do that, it is crucial to combine during the search the maintaining of good diversity around the complete domain with an intensification in the most promising regions [229]. Thus, a tendency is to introduce better operations to enforce one or both criteria to improve the behavior of metaheuristics [130], [230].

The population is not the only memory structure used in metaheuristics algorithms. Sometimes, additional memory structures are used to improve the search. Initially, these memory structures were mainly used by algorithms especially designed to used them, as Tabu Search [17] (for avoid cycles) or Scatter Search [108] (to have diverse solutions to introduce diversity). Nowadays, by contrast, the incorporation of an additional memory that store solutions or other information obtained during the search is a new trend. The memory structure can be used to store the decisions chosen [17] (to avoid cycles), to store the current state (to continue in a near future [92]) or to measure the results obtained by each component (as the local search or the exploration algorithm applied) to adapt its application. Measuring and comparing the performance obtained by each component allow to have a set of different techniques and to apply the most promising one

in each time, using self-adaptive parameters [79], [231]. An archive of generated solutions can maintain a growing set of good solutions to better guide the search [80]. This is specially interesting in multiobjective optimization and multimodal optimization: In multiobjective optimization to store the nondominated solutions to improve the optimization [160], [232]. In multimodal optimization, the number of optima could be many times greater than the population size, thus an additional memory storing found solutions allows the algorithm to remove them from the population to favor the exploration [233], [96], [234].

One aspect that it is a discussion topic is the huge number of nature-inspired metaheuristics. Unfortunately, some of them are *innovative* with respect to the natural element that has inspired it but they are not really innovative in its behavior and they do not offer a real improvement over previous algorithms. There are algorithms very well inspired from nature (as PSO) but there are others that do not need this *natural* origin to obtain good results (as DE). Thus, this explosion of nature-inspired algorithms is currently the object of debate [235], [236].

Recently, there is a discussion about two different ways of focusing the optimization problem. One option is the design of metaheuristics that try to obtain good results in a group of problem, and the other is to chose an real problem, and try to obtain the best metaheuristic for that problem. Each option has its advantages and drawbacks: On one hand, it is true that metaheuristics can improve existing algorithms in specific problems [237] and that solving real engineering problem can have a great interest for companies, but the resulting algorithms sometimes are too specific to them and not easily adaptable for new problems (the reusability of the effort is reduced); on the other hand, the design of algorithms with a robust behavior in many problems is the ideal but not only they can get worse results on specific real-world problems but also they are tested using benchmarks that contain many functions difficult to optimize but with no interest in real-world [238], [239].

Another important topic of debate is whether the ideal aim in optimization with metaheuristics, to have an efficient and parameter-free algorithm to solve different problems (as much as possible), is possible to achieve it. The theory of non-free lunch [240] denies that possibility, indicating that *any two algorithms are equivalent when their performance is averaged across all possible problems.* This theory implies that there would never be an algorithm statistically better than the other, because although it could be better than others in a group of functions, it should be worse in others, being similar in average. However, new studies shows that in the field of real-world problems this theory could not be applied [241].

Empirical results appears to support, at least partially, the previous affirmation showing a great difficulty in getting an optimization algorithm that improves the others in the majority of functions. In response to that, an increasing number of researchers propose hybrid metaheuristics, that combine by hybridization different algorithms that could work well together, to increase the types of functions in which the algorithm has a good behavior. Initially, the hybridizations were mainly MAs adding an exploitation algorithm to a global exploration algorithm to improve the accuracy, but in recent years the variety of algorithms used for hybridization is overwhelming: PSO s with DEs [140], ACOs with CMA-ES [95], … Another important change in new hybrid metaheuristics is how the different algorithms are used. Instead of applying the different algorithms during each iteration, several of them apply each time only one of the component algorithms: one at the beginning and other at the end of the running, or select alternative which should be use during the run (usually with an self-adaptive criterion) [79]. Withit this tendency, the term memetic computing, (MC), has been proposed as as a paradigm that uses the notion of meme(s) as any units of information encoded in computational representations for the purpose of problem-solving. Thus, MC is a more general concept than MA and it can include the new hybrid metaheuristics [242], [243], [244].

10. Conclusion

This paper gives a walk into the principles and the state-of-the-art of metaheuristic methods for solving complex, nonlinear engineering optimization problems. All the techniques discussed are classified into three basic types: trajectory based approaches, multi-trajectory based approaches, and population based approaches, depending on whether a single point or multiple points are generated as new approximate solution(s) in each step of the iterations. Generally, the trajectory based approaches are simple and effective for optimization problems with a low number of parameters. However, when the dimension of the space increases, they become less efficient and are more likely to get stuck in a local optimum. In many practical applications, a multi-trajectory based search method is used, combining the trajectory based search with a multi-start strategy to increase the chance to find a global optimum. The population based approaches are superior to the

trajectory based ones in global search capability; they seem to be more suitable to be applied in high-dimensional search spaces. But, larger memory requirements and more computational cost are connected with them as side effects.

Secondly, we have discussed three issues especially interesting: multimodal optimization in which we are interested in obtaining several optima; multi-objective optimization that aims to find solutions to minimize/maximize several objectives at the same time; as well as parallel design and implementation of metaheuristic algorithms. Also we have reviewed the available software frameworks for metaheuristics, considering their different features.

Later, we have remarked several issues that are considered as crucial challenges for application of metaheuristics in complex engineering scenarios: large scale optimization, i.e., optimization of solutions with high dimensionality; optimization using a very small number of evaluations in cases of expensive (in time) fitness functions; and the challenge of alleviating parameters in metaheuristic algorithms, using automatic tuning or self-adaption of parameters.

Finally, we have presented several relevant discussion topics that that are important for the success of metaheuristics and discussed in the literature: the influence of a good balance between the exploration and exploitation; how an additional memory can be used to improve results; the importance of a nature inspiration for a metaheuristic; the alternative of metaheuristics designed for specific problems or designed for more general problems; the influence of the *non-free lunch* theory over optimization, and new tendencies in hybrid metaheuristics.

Acknowledgement

The work is within the EMOPAC project (project no 16317) granted by the Swedish Knowledge Foundation. We are also grateful to ABB FACTS, Prevas, and VG Power for co-financing the research. This work was supported in part by the Spanish Ministry of Education and Science under Grant TIN2011-28488 and TIN 2012-37930-C02-01 and the Andalusian Government under Grant P10-TIC-6858.

References

1. L. Shi, S. Olafsson and Q. Chen, An optimization framework for product design, *Management Science* **47** (2001) 1681-1692.
2. M. S. Bazaraa, J. J. Jarvis, and H. D. Sherali, *Linear Programming and Network Flows,* 2nd edn. (John Wiley & Sons, New York, 1990).
3. D. G. Luenberger, *Linear and Non-linear Programming* (Addison-Wesley, New York, 1990).
4. M. S. Bazaraa, H. D. Sherali, and C. M. Shetty, *Nonlinear Programming – Theory and Algorithms* (John Wiley & Sons, New York, 1993).
5. L. G. Khachian, A polynomial algorithm in linear programming, *Soviet Mathematics Doklady* **20** (1979) 1093-1096.
6. N. Karmarkar, A new polynomial algorithm for linear programming, *Combinatorica* **4** (1984) 373-395.
7. W. H. Swann, A survey of non-linear optimization techniques, *FEBS Letters* **2** (1969) 39-55.
8. I. H. Osman and G. Laporte, Metaheuristics: A bibliography, *Annals of Operations Research* **63** (1996) 513-562.
9. F. Glover and G. A. Kochenberger (Eds.), *Handbook of metaheuristics*, International Series in Operations Research & Management Science (Springer, 2003).
10. I. Boussaïd, J. Lepagnot, P. Siarry, A survey on optimization metaheuristics, Information Sciences, **237** (2013) 82-117.
11. S. Russel and P. Norvig, *Artificial Intelligence: A Modern Approach*, 2nd edn. (Prentice Hall, New Jersey, 2003).
12. S. B. Kjær, Evaluation of the "Hill Climbing" and the "Incremental Conductance" maximum power point trackers for photovoltaic power systems, *IEEE Transactions on Energy Conversion* **27** (4) (2012) 922–929.
13. K. A. Sullivan and S. H. Jacobson, Ordinal hill climbing algorithms for discrete manufacturing process design optimization problems, *Discrete Event Dynamic Systems* **10** (2000) 307-324.
14. S. Kirkpatrick, C. D. Gelatt and M. P. Vecchi, Optimization by simulated annealing, *Science* **220** (1983) 671-680.
15. P. Van Laarhoven and E. Aarts, *Simulated Annealing: Theory and Applications* (Kluwer Academic Publishers, Norwell, 1987).
16. F. Glover, Tabu search – Part one, *ORSA Journal Computing* **1** (1989) 190-206.
17. F. W. Glover and M. Laguna, *Tabu Search* (Springer, 1997).
18. R. Chelouah and P. Siarry, Tabu Search applied to global optimization. European Journal of Operational Research, **123** (2) (200) 256-270.
19. J. Blazewicz, P. Lukasiak and M. Milostan, Application of tabu search strategy for finding low energy structure of protein, *Artificial Intelligence in Medicine* **35** (1-2) (2013) 135-145.
20. R. E. Aleman and R. R. Hill, A tabu search with vocabulary building approach for the vehicle routing problem with split demands, *Journal of Metaheuristics* **1** (1) (2010) 55-80.

21. M. Avriel, *Nonlinear Programming: Analysis and Methods* (Dover Publishing, 2003).
22. R. Battiti, First- and second-order methods for learning: Between steepest descent and Newton's method, *Neural Computation* **4** (1992) 141-166.
23. J. F. Bonnans, J. Ch. Gilbert, C. Lemaréchal and C. A. Sagastizábal, *Numerical Optimization, Theoretical and Numerical Aspects*, 2nd edn. (Springer, 2006).
24. J. A. Nelder and R. A. Mead, A simplex for function minimization, *Computer Journal* **7** (1965) 308-313.
25. J. S. Arora, O. A. Elwakeil and A. I. Chahande, Global optimization method for engineering applications: a review, *Structural Optimization* **9** (1995) 137-159
26. R. Helena, O. M. Lourenço and T. Stützle, Iterated local search, in *Handbook of Metaheuristics* (Springer, 2003), pp. 320-353.
27. N. Mladenovic and P. Hansen, Variable neighborhood search, *Computers and Operations Research* **24**(11) (1997) 1097-1100.
28. P. Hansen, N. Mladenovic and J. A. M. Perez, Variable neighborhood search: Methods and applications, *Annals of Operation Research* **175** (2010) 367-407.
29. T. A. Feo and M. G. C. Resende, Greedy randomized adaptive search procedures, *Journal of Global Optimization* **6** (1995) 109-134.
30. L. Pitsoulis and M. G. C. Resende, Greedy randomized adaptive search procedures, in *Handbook of Applied Optimization* (Oxford University Press, 2002), pp. 168-181.
31. M. Prais and C. C. Ribeiro, Reactive GRASP: An application to a matrix decomposition problem in TDMA traffic assignment, *INFORMS Journal on Computing* **12** (2000) 164-176.
32. J.C. Culberson, and F. Luo, Exploring the k-colorable landscape with iterated greedy, Dimacs Series in Discrete Mathematics and Theoretical Computer Science, American Mathematical Society (1996) pp. 245-284.
33. R. Ruiz, T. Stützle, A simple and effective iterated greedy algorithm for the permutation flowshop scheduling problem, *European Journal of Operational Research*, **177**(3) (2007) 2033–2049.
34. K.C Ying and M.H. Cheng, Dynamic parallel machine scheduling with sequence-dependent setup times using an iterated greedy heuristic, *Expert Systems with Applications,* **37**(4), (2010), 2848-2852.
35. M. Lozano, D. Molina and García-Martínez, Iterated greedy for the maximum diversity problem, *European Journal of Operational Research*, **214**(1) (2011) 31-38.
36. G. Minella, R. Ruiz and M. Ciavotta, Restarted Iterated Pareto Greedy algorithm for multi-objective flowshop scheduling problems, *Computers and Operational Research* **38**(11) (2011) 1521-1533.
37. F.J. Rodríguez, M. Lozano, C. Blum and C. García-Martínez, iterated greedy algorithm for the large-scale unrelated parallel machines scheduling problem, *Computers and Operational Research* **40**(7) (2013) 1829-1841.
38. R. A. Formato, Central force optimization: A new metaheuristic with applications in applied electromagnetics, *Progress in Electromagnetics Research* **77** (2007) 425-491.
39. P. K. Roy, B. Mandal and K. Bhattacharya, Gravitational search algorithm based optimal reactive power dispatch for voltage stability enhancement, *Electronic Power Components and Systems* **9**(9) (2014) 956-976.
40. A. Hatamlou, Black hole: A new heuristic optimization approach for data clustering, *Information Science* **222** (2013) 175-184.
41. Z. W. Geem, J. H. Kim and G. V. Loganathan, A new heuristic optimization algorithm: Harmony search, *Simulation* **76**(2) (2001) 60-68.
42. Y. Lui and K. M. Passino, Biomimicry of social foraging bacteria for distributed optimization: Models, principles, and emergent behaviors, *Journal of Optimization Theory and Applications* **115**(3) (2002) 603-628.
43. L. N. de Castro and J. I. Timmis, Artificial immune systems as a novel soft computing paradigm, *Soft Computing* **7** (8) (2003) 526-544.
44. D. Simon, Biogeography-based optimization, *IEEE Transactions on Evolutionary Computation* **12** (2008) 702-713.
45. A. Puris, R. Bello, D. Molina and F. Herrera, Variable mesh optimization for continuous optimization problems*, Soft Computing* **16** (2012) 511-525.
46. K. N. Krishnanand and D. Ghose, Glowworm swarm optimization for simultaneous capture of multiple local optima of multimodal functions, *Swarm Intelligence* **3**(2) (2009) 87-124.
47. X.S. Yang and S. Deb, Engineering Optimisation by Cuckoo Search, *Int. J. Mathematical Modelling and Numerical Optimisation* **1** (4) (2010) 330-343.
48. X. Li, J. Luo, M.-R. Chen and N. Wang, An improved shuffled frog-leaping algorithm with external optimisation for continuous optimisation, *Information Sciences* **192** (2012) 143-151.
49. D. E. Goldberg, *Genetic Algorithms in Search, Optimization and Machine Learning* (Addison-Wesley, New York, 1989).
50. D. E. Goldberg, *The Design of Innovation: Lessons from and for Competent Genetic Algorithms* (Kluwer Academic Publishers, Norwell, MA, USA, 2002).
51. F. Herrera, M. Lozano and J. L. Verdegay, Tackling real-coded genetic algorithms: Operators and tools for behavioral analysis, *Artificial Intelligence Review* **12** (1998) 265-319.
52. J. R. Koza, *Genetic Programming: On the Programming of Computers by Means of Natural Selection* (*Complex Adaptive Systems*) (The MIT Press, 1992).
53. R. I. McKay, N. X. Hoai, P. A. Whigham and Y. S. M. O'Neill, Grammar-based genetic programming: A survey, *Genetic Programming and Evolvable Machines* **11** (2010) 365–396.
54. B. L. William, P. Riccardo, F. M. Nicholas and R. K. John, Genetic programming: An introduction and tutorial, with a survey of techniques and applications, in *Computational Intelligence: A Compendium*, Studies in Computational Intelligence (SCI) (Springer-Verlag, 2008), pp. 927–1028.
55. R. Poli, W. B. Langdon and N. F. McPhee, *A Field Guide to Genetic Programming* (Lulu Enterprises, UK Ltd., 2008).

56. A. Piszcz and T. Soule, A survey of mutation techniques in genetic programming, in *Proc. 8th Annual Conf. Genetic and Evolutionary Computation*, GECCO '06 (New York, 2006), pp. 951–952.
57. I. Rechengerg, *Evolutionsstrategie – Optimierung technischer Systeme nach Prinzipien der biologischen Evolution* (PhD thesis) (Fromman-Holzboog, 1973).
58. N. Hansen and A. Ostermeier, Completely derandomized self-adaptation in evolution strategies, *Evolutionary Computation* **9** (2) (2001) 159-195.
59. T. Liao, A. M. M. de Oca, and T. Stützle, Tuning parameters across mixed dimensional instances: A performance scalability study of Sep-G-CMA-ES, in *Proc. the 13th Annual Conf. Genetic and Evolutionary Computation (GECCO '11)* (New York, 2011), pp. 703-706.
60. A. Auger and N. Hansen, A restart CMA evolution strategy with increasing population size, in *Proc. IEEE Congress on Evolutionary Computation* (2005), pp. 1769-1776.
61. N. Hansen, Benchmarking a BI-Population CMA-ES on the BBOB function testbed, in *Proc. Genetic and Evolutionary Computation Conference* (2009), pp. 2389-2396.
62. T. Liao and T. Stutzle, Benchmark results for a simple hybrid algorithm on the CEC 2013 benchmark set for real-parameter optimization, in *Proc. IEEE Congress on Evolutionary Computation* (2013), pp. 1938-1944.
63. L. J. Fogel, A. J. Owens and M. J. Walsh, *Artificial Intelligence through Simulated Evolution* (John Wiley, New York, USA, 1966).
64. D. B. Fogel, L. J. Fogel, J. Atma and G. Fogel, Hierarchic methods of evolutionary programming, in *Proc. the First Annual Conf. Evolutionary Programming* (La Jolla, CA, 1992), pp. 175–182.
65. X. Yao, Y. Liu and G. Lin, Evolutionary programming made faster, *IEEE Transactions on Evolutionary Computation* **3** (1999) 82–102.
66. C. Y. Lee and X. Yao, Evolutionary programming using mutations based on the levy probability distribution, *IEEE Transactions on Evolutionary Computation* **8** (2004) 1–13.
67. H. Mühlenbein and G. Paaß, From recombination of genes to the estimation of distributions in binary parameters, In *Lecture Notes in Computer Science 1411: Parallel Problem Solving from Nature - PPSN IV* (Springer,1996), pp 178-187.
68. P. Larrañaga and J.A. Lozano, *Estimation of Distribution Algorithms: A New Tool for Evolutionary Computation* (Kluwer Academic Publishers, 2001).
69. M. Pelikan, D. E. Golderg and F. G. Lobo, A survey of optimization by building and using probabilistic models, *Computational Optimization and Application* **21** (2002) 5-22.
70. M. Hauschild and M. Pelikan, An introduction and survey of estimation of distribution algorithms, *Swarm and Evolutionary Computation* **1** (2011) 111-128.
71. J. Ceberio, E. Irurozki, A. Mendiburu and J.A. Lozano, A review on estimation of distribution algorithms in permutation-based combinatorial optimization problems, *Progress in Artificial Intelligence* **1**(1) (2012) 103-117.
72. R. Armañanzas, I. Inza, R. Santana, Y. Saeys, J. L. Flores, J. A. Lozano, Y. Van de Peer, R. Blanco, V. Robles, C. Bielza and P. Larrañaga, A review of estimation of distribution algorithms in bioinformatics, *BioData Mining* **1**(6) (2008).
73. J. Kennedy and R. Eberhart, Particle swarm optimization, in *Proc. IEEE Conf. Neural Networks* (1995), pp. 1942–1948.
74. J. Kennedy, R. C. Eberhart and Y. Shi, *Swarm Intelligence* (Morgan Kaufman, San Francisco, USA, 2001).
75. W.-N. Chen et al., A novel set-based particle swarm optimization method for discrete optimization problems, *IEEE Transactions on Evolutionary Computation* **14**(2) (2010) 278–300
76. J. J. Liang, A. K. Qin, P. N. Suganthan and S. Baskar, Comprehensive learning particle swarm optimizer for global optimization of multimodal functions, *IEEE Transactions on Evolutionary Computation* **10**(3) (2006) 281–295.
77. R. Storn and K. Price, Differential evolution - A simple and efficient heuristic for global optimization over continuous spaces, *Journal of Global Optimization* **11** (1997) 341–359.
78. K. V. Price, R. M. Rainerm and J. A. Lampinen, *Differential Evolution: A Practical Approach to Global Optimization* (Springer-Verlag, 2005).
79. A. K. Qin, V. L. Huang and P. N. Suganthan, Differential evolution algorithm with strategy adaptation for global numerical optimization, *IEEE Transactions on Evolutionary Computation* **13**(2) (2002) 398–417.
80. J. Zhang and A. C. Sanderson, JADE: Adaptive differential evolution with optional external archive, *IEEE Transactions on Evolutionary Computation* **13**(5) (2009) 945–958.
81. S. Das and P. N. Suganthan, Differential evolution: A survey of the state-of-the-art, *IEEE Transactions on. Evolutionary Computation* **15** (2011) 4-31.
82. D. Karaboga and B. Basturk, A powerful and efficient algorithm for numerical function optimization: artificial bee colony (ABC) algorithm, *Journal of Global Optimization* **39**(3) (2007) 459–471.
83. D. Karaboga, B. Gorkemli, C. Ozturk and N. Karaboga, A comprehensive survey: artificial bee colony (ABC) algorithm and applications, *Artificial Intelligence Review* **42**(1) (2012) 21–57.
84. P. Moscato, *On Evolution, Search, Optimization, Genetic Algorithms and Martial Arts: Towards Memetic Algorithms*, Caltech Concurrent Computation Program Report 826 (1989).
85. N. Krasnogor and J. Smith, A tutorial for competent memetic algorithms: Model, taxonomy, and design issues, *IEEE Transaction on Evolutionary Computation* **9**(5) (2005) 474-488.
86. J. Du, R. Rada, Memetic algorithms, domain knowledge, and financial investing, *Memetic Computing,* **4**(2) (2012) 109-125.
87. Y. S. Ong and A. J. Keane, Meta-Lamarckian in memetic algorithm, *IEEE Transactions on Evolutionary Computation* **8** (2) (2004) 99-110.
88. E. G. Talbi, A taxonomy of hybrid metaheuristics, *Journal of Heuristics* **8**(5) (2002), 541-564.
89. P. Merz, Advanced fitness landscape analysis and the performance of memetic algorithms, *Evolutionary Computation* **12**(3) (2004) 303–325.

90. C. Blum, J. Puchinger, G. R. Raidl and A. Roli, Hybrid metaheuristics in combinatorial optimization: A survey, *Applied Soft Computing* **11**(6) (2011) 4135–4151.
91. Q. H. Nguyen, Y.-S. Ong and N. Krasnogor, A study on the design issues of memetic algorithm, in *Proc. IEEE Congress on Evolutionary Computation* (2007), pp. 2390–2397.
92. D. Molina, M. Lozano, C. Garcia-Martinez and F. Herrera, Memetic algorithms for continuous optimization based on local search chains, *Evolutionary Computation* **18**(1) (2010) 27-63.
93. J. J. Liang, B-Y. Qu, P. N. Suganthan and A. G. Hernández-Díaz, *Problem definitions and evaluation criteria for the CEC 2013 special session and competition on real-parameter optimization*, Technical Report 201212, Computational Intelligence Laboratory, Zhengzhou University, Zhengzhou, China and Technical Report, Nanyang Technological University, Singapore (2013).
94. I. Loshchilov. CMA-ES with restarts for solving CEC 2013 benchmark problems, in *Proc. IEEE Congress on Evolutionary Computation* (2013), pp. 369–376.
95. T. Liao and T. Stutzle, Benchmark results for a simple hybrid algorithm on the CEC 2013 benchmark set for real-parameter optimization, in Proc. *IEEE Congress on Evolutionary Computation* (2013), pp. 1938-1944.
96. B. Lacroix, D. Molina and F. Herrera, Dynamically updated region based memetic algorithm for the 2013 CEC special session and competition on real parameter single objective optimization, in *Proc. IEEE Congress on Evolutionary Computation* (2013), pp. 1945–1951.
97. J. J. Liang, B-Y. Qu and P. N. Suganthan, *Problem definitions and evaluation criteria for the CEC 2014 special session and competition on single objective real-parameter numerical optimization*, Technical Report 201311, Computational Intelligence Laboratory, Zhengzhou University, Zhengzhou, China and Technical Report, Nanyang Technological University (2013).
98. S. Elsayed, S. Ruhul, D. Essam and N. Hamza, Testing united multi-operator evolutionary algorithms on the CEC2014 real-parameter numerical optimization, in *Proc. IEEE Congress on Evolutionary Computation* (2014), pp. 1650-1657.
99. R. Tanabe and A. Fukunaga, Improving the search performance of SHADE using linear population size reduction, in *Proc. IEEE Congress on Evolutionary Computation* (2014), pp. 1658-1665.
100. I. Erlich, J. L. Rueda and S. Wildenhues, Evaluating the mean-variance mapping optimization on the IEEE-CEC 2014 test suite, in *Proc. IEEE Congress on Evolutionary Computation* (2014), pp. 1625-1632.
101. F. Neri and C Cotta, Memetic algorithms and memetic computing optimization: A literature review, *Swarm and Evolutionary Computation* **2** (2012) 1–14.
102. M. Dorigo, V. Maniezzo and A. Colorni, The ant system: Optimization by a colony of cooperating agents, *IEEE Transactions on Systems, Man, and Cybernetics – Part B* **26** (1996) 29-41.
103. M. Dorigo and T. Stützle, *Ant Colony Optimization* (MIT Press, 2004).
104. M. Dorigo and L. M. Gambardella, Ant colony system: A cooperative learning approach to the traveling salesman problem, *IEEE Transactions on Evolutionary Computation* **1** (1997) 53-66.
105. T. Liao, T. Stützle, M. A. Montes de Oca and M. Dorigo, A unified ant colony optimization algorithm for continuous optimization, *European Journal of Operational Research* **234**(3) (2014) 597–609.
106. B. Chandra Mohan and R. Baskaran, A survey: ant colony optimization based recent research and implementation on several engineering domains, *Expert Systems with Applications* **39**(4) (2012) 4618–4627.
107. F. Glover, A template for scatter search and path relinking, *Lecture Notes on Computer Science* 1363 (1997), pp.13-54.
108. M. Laguna and R. Martí, *Scatter Search: Methodology and Implementations in C* (Springer, 2013).
109. J. G. Villegas, C. Prins, C. Prodhon, A. L. Medaglia and N. Velasco, A GRASP with evolutionary path relinking for the truck and trailer routing problem, *Computers and Operations Research* **38**(9) (2011) 1319-1334.
110. V. A. Armentano, A. L. Shiguemoto and A. Løkketangen, Tabu search with path relinking for an integrated production distribution problem, *Computer and Operational Research* **38** (2011) 1199-1209.
111. K. A. De Jong, An analysis of the behavior of a class of genetic adaptive systems, Doctoral Dissertation (Comut. Commun. Sci., University Michigan, Ann Arbor, MI, 1975).
112. S. W. Mahfoud, Niching methods for genetic algorithms, Ph.D. Dissertation (Univ. of Illinois, Urbana-Champaign, 1995).
113. C. H. Lee, D. Cho and H. Jung, Niching genetic algorithm with restricted competition selection for multimodal function optimization, *IEEE Transactions on. Magnetics* **35**(3) (1999) 1722-1725.
114. J. Holland, *Adaptation in Natural and Artificial Systems* (University of Michigan Press, Ann Arbor, MI, 1975).
115. D. E. Goldberg and J. Richardson, Genetic algorithms with sharing for multimodal function optimization, in *Proc. 2nd Int. Conf. Genetic Algorithms* (1987), pp. 41-49.
116. A. Petrowski, A clearing procedure as a niching method for genetic algorithm, in *Proc. IEEE Conf. Evolutionary Computation* (Japan, 1996), pp. 798-803.
117. J. Li, M. Balazs, G. T. Parks and P. J. Clarkson, A species conserving genetic algorithm for multimodal function optimization, *Evolutionary Computation* **10**(3) (2002) 207-234.
118. X. Li, Efficient differential evolution using speciation for multimodal function optimization, in *Proc. Conf. Genetic and Evolutionary Computation* (2005), pp. 873-880.
119. S. Yang and C. Li, A clustering particle swarm optimizer for locating and tracking multiple optima in dynamic environments, *IEEE Transactions on Evolutionary Computation* **14**(6) (2010) 959-974.
120. T. Kanungo, D. M. Mount, N. S. Netanyahu, C. D. Piatko, R. Silverman and A. Y. Wu, An Efficient k -Means clustering algorithm: Analysis and implementation, *IEEE Transactions*

on Pattern Analysis and Machine Intelligence **24**(7) (2002) 881–892.
121. B. Sareni and L. Krähenbühl, Fitness sharing and niching methods revisited, *IEEE Transactions on Evolutionary Computation* **2**(3) (1998) 97–106.
122. C. Lin and W. Wu, Niche identification techniques in multimodal genetic search with sharing scheme, *Advances in Engineering Software* **33** (2002) 779-791.
123. A. D. Cioppa, C. De Stefano and A. Marcelli, Where are the niches? Dynamic fitness sharing, *IEEE Transactions on Evolutionary Computation* **11** (2007) 453–465.
124. G. Dunwey, P. Fengping and X. Shifan, Adaptive niche hierarchy genetic algorithm, in Proc. IEEE Conf. TENCON (2002), pp. 39–42.
125. E. Perez, M. Posada and F. Herrera, Analysis of new niching genetic algorithms for finding multiple solutions in the job shop scheduling, *Journal of Intelligent Manufacturing* **23** (2012) 341-256.
126. E. L. Yu and P. N. Suganthan, Ensemble of niching algorithm, *Information Sciences* **180**(15) (2010) 2815–2833.
127. E. Perez, M. Posada and F. Herrera, Analysis of new niching genetic algorithms for finding multiple solutions in the job shop scheduling, *Journal of Intelligent Manufacturing* **23** (2012) 341-256.
128. R. Thomsen, Multimodal optimization using crowding-based differential evolution, in *Proc. IEEE Congress on Evolutionary Computation* (2004), pp. 1382-1389.
129. S. Kundu, S. Biswas, S. Das and P. N. Suganthan, Crowding-based local differential evolution with speciation-based memory archive for dynamic multimodal optimization, in *Proc. Conf. Genetics and Evolutionary Computation* (2013), pp. 33-40.
130. B. Y. Qu, P. N. Suganthan and J. J. Liang, Differential evolution with neighborhood mutation for multimodal optimization, *IEEE Transactions on Evolutionary Computation* **16** (5) (2012) 601-614.
131. J. Li, M. Balazs, G. T. Parks and P. J. Clarkson, A species conserving genetic algorithm for multimodal function optimization, *Evolutionary Computation* **10**(3) (2002) 207-234.
132. X. Li, Efficient differential evolution using speciation for multimodal function optimization, in *Proc. Conf. Genetic and Evolutionary Computation* (2005), pp. 873-880.
133. D. Parrott and X. Li, Locating and tracking multiple dynamic optima by a particle swarm model using speciation, *IEEE Transactions on Evolutionary Computation* **10**(4) (2006) 440-458.
134. M. G. Epitropakis, X. Li and E. K. Burke, A dynamic archive niching differential evolution algorithm for multimodal optimization, in *Proc. IEEE Congress on Evolutionary Computation* (Cancun, Mexico, 2013), pp. 79-86.
135. X. Li, A. Engelbrecht and M.G. Epitropakis, Benchmark functions for CEC'2013 special session and competition on niching methods for multimodal function optimization, Technical Report (Evolutionary Computation and Machine Learning Group, RMIT University, Australia, 2013).
136. R. Brits, A. P. Engelbrecht and F. van den Bergh, Locating multiple optima using particle swarm optimization, *Applied Mathematics and Computation* **189**(2) (2007) 1859–1883.
137. P.-Y. Yin et al., Niched ant colony optimization with colony guides for QoS multicast routing, *Journal of Network and Computer Applications* (2014), in press.
138. O. M. Shir and T. Baeck, Niche radius adaptation in the CAM-ES niching algorithm, in *Proc. Conf. Parallel Problem Solving from Nature* (Springer-Verlag, Berlin, Germany, 2006), pp. 142-151.
139. M. Peruss, Niching the CMA-ES via nearest-better clustering, in *Proc. Conf. Genetic and Evolutionary Computation* (Portland, USA, 2010), pp. 1711-1717.
140. B. Xin, J. Chen, J. Zhang, H. Fang and Z.-H. Peng, Hybridizing differential evolution and particle swarm optimization to design powerful optimizers: A review and taxonomy, *IEEE Transactions on Systems, Man, and Cybernetics, Part C* **42**(5) (2012) 744 – 767.
141. K. Deb and A. Saha, Finding multiple solutions for multimodal optimization problems using a multi-objective evolutionary approach, in Proc. 12th Annual Conf. Genetic and Evolutionary Computation (USA, 2010), pp. 447–454.
142. K. Miettinen, *Nonlinear Multiobjective Optimization* (Kluwer, Norwell, MA, 1999).
143. H. Ishibuchi and T. Murata, Multi-objective genetic local search algorithm and its application to flowshop scheduling, *IEEE Transactions on Systems, Man, & Cybernetics* **28** (1998) 392-403.
144. A. Jaszkiewicz, On the performance of multiple-objective genetic local search of the 0/1knapsack problem – A comparative experiment, *IEEE Transactions on Evolutionary Computation* **6** (4) (2002) 402-412.
145. Q. Zhang and H. Li, MOEA/D: A multiobjective evolutionary algorithm based on decomposition, *IEEE. Transactions on Evolutionary Computation* **11**(6) (2007) 712-731.
146. Q. Zhang, W. Liu and H. Li, The Performance of a New version of MOEA/D on CEC09 unconstrained MOP test instances, Working Report CES-491 (School of Computer Science and Electrical Engineering, University of Essex, 02/2009).
147. Q. Zhang, A. Zhou, S. Zhao, P. N. Suganthan, W. Liu and S. Tiwari, Multiobjective optimization test instances for the CEC 2009 special session and competition, Working Report CES-887 (School of Computer Science and Electrical Engineering, University of Essex, 2008).
148. H. Li and Q. Zhang. Multiobjective optimization problems with complicated Pareto sets, MOEA/D and NSGA-II, *IEEE Transactions on Evolutionary Computation* **12**(2) (2009) 284-302.
149. L. Keng, Q. Zhang and R. Battiti, MOEA/D-ACO: A multiobjective evolutionary algorithm using decomposition and ant colony, *IEEE Transactions on Cybernetics* **43**(6) (2013) 1845-1859.
150. J. J. Durillo, Q. Zhang, A. J. Nebro and E. Alba, Distribution of computational effort in parallel MOEA/D, in *Lecture Notes in Computer Science*, Volume 6683 (Springer, 2011), pp 488-502.

151. C. M. Fonseca and P. J. Fleming, Multiobjective optimization and multiple constraint handling with evolutionary algorithms, part I: A unified formulation, *IEEE. Transactions on Systems, Man, & Cybernetics, Part A* **28** (1998) 26-37.
152. N. Srinivas and K. Deb, Multiobjective function optimization using nondominated sorting genetic algorithm, *Evolutionary Computation* **2** (1995) 221-248.
153. E. Zitzler and L. Thiele, Multiobjective evolutionary algorithms: A comparative case study and the Strength Pareto Approach, *IEEE Transactions on Evolutionary Computation* **3** (4) (1999) 257-271.
154. E. Zitzler, M. Laumanns and L. Thiele, *SPEA2: Improving the Strength Pareto Evolutionary Algorithm*, Technical Report (Computer Engineering and Networks Laboratory (TIK), Swiss Federal Institute of Technology, 2001).
155. K. Deb, A. Pratap, S. Agrawal and T. Meyarivan, A fast and elitist multiobjective genetic algorithm: NSGA-II, *IEEE Transactions on Evolutionary Computation* **6** (2002) 182-197.
156. K. Deb and H. Jain. An evolutionary many-objective optimization algorithm using reference-point-based nondominated sorting approach, part I: Solving problems with box constraints. *IEEE Transactions on Evolutionary Computation* **18**(4) (2014) 577-601.
157. H. Jain and K. Deb. n Evolutionary Many-Objective Optimization Algorithm Using Reference-Point-Based Nondominated Sorting Approach, Part II: Handling Constraints and Extending to an Adaptive Approach. *IEEE Transactions on Evolutionary Computation* **18**(4) (2014) 602-622.
158. D. S. Liu, K. C. Tan, C. K. Huang, C. K. Goh and W. K. Ho, On solving multiobjective bin packing problems using evolutionary particle swarm optimization, *European Journal of Operational Research* **190** (2008) 357-382.
159. P. K. Tripathi, S. Bandyopadhyay and S. K. Pal, Multi-objective particle swarm optimization with time variant inertia and acceleration coefficients, *Information Sciences* **177** (2007) 5033-5049.
160. N. K. Madavan, Multiobjective optimization using a Pareto differential evolution approach, in *Proc. IEEE Congress on Evolutionary Computation* (2002), pp. 1145-1150.
161. F. Xue, A. C. Sanderson and R. J. Graves, Pareto-based multi-objective differential evolution, in *Proc. IEEE Congress on Evolutionary Computation* (2003), pp. 862-869.
162. T. Robic and B. Filipic, DEMO: Differential evolution for multiobjective optimization, in *Proc. 3rd Int. Conf. Evolutionary Multicriterion Optimization* (2005), pp. 520-533.
163. M. Ali, P. Siarry and M. Pant, An efficient differential evolution based algorithm for solving multi-objective optimization problems, *European Journal of Operational Research* **217** (2012) 404-416.
164. R. Sarker and H. Abbass, Differential evolution for solving multiobjective optimization problems, *Asia Pacific Journal of Operational Research* **21** (2) (2004) 225-240.
165. W. Gong and Z. Cai, An improved multiobjective differential evolution based on Pareto-adaptive epsilon-dominance and orthogonal design, *European Journal of Operational Research* **198** (2) (2009) 576-601.
166. E. Alba, G. Luque and S. Nesmachnow, Parallel metaheuristics: recent advances and new trends, *International Transactions in Operational Research* **20**(1) (2013) 1-48.
167. E. Alba, *Parallel Metaheuristics: A New Class of Algorithms* (Wiley, New York, 2005).
168. E. Talbi, *Parallel Combinatorial Optimization* (Wiley, 2006).
169. F. Caraffini, F. Neri, G. Iacca and A. Mol, Parallel memetic structures, *Information Sciences* **227** (2013) 60-82.
170. A. Mendiburu, J. Miguel-Alonso and J.A. Lozano, A review of parallel estimation of distribution algorithms, in *Studies in Computational Intelligence, Parallel and Distributed Computational Intelligence* (Springer, Berlin-Heidelberg, 2010).
171. V. O. Shylo, T. Middelkoop and P. Pardalos, Restart strategies in optimization: parallel and serial cases, *Parallel Computing* **37**(1) (2011) 60-68.
172. R. Tanese, Distributed genetic algorithms, in *Proc. 3rd Int. Conf. Genetic Algorithms* (1989), pp. 434-439.
173. J. Lampinen, Differential evolution: New naturally parallel approach for engineering design optimization, in *Developments in Computational Mechanics with High Performance Computing* (Civil-Comp Press, 1999), pp. 217-228.
174. A. P. Piotrowski, J. J. Napiorkowski and A. Kiczko, Differential evolution algorithm with separated groups for multi-dimensional optimization problems, *European Journal of Operational Research* **216**(1) (2012) 33-46.
175. J. F. Schutte, J. A. Reinbolt, B. J. Fregly, R. T. Haftka and A. D. Georg, Parallel global optimization with the particle swarm algorithm, *International Journal for Numerical Methods in Engineering* **61**(13) (2004) 2296-2315.
176. M. Hijaze and D. Corne, An Investigation of topologies and migration schemes for asynchronous distributed evolutionary algorithms, in *Proc. World Congress on Nature & Biologically Inspired Computing* (2009), pp. 636-641.
177. M. Ruciński, D. Izzo and F. Biscani, On the impact of the migration topology on the island model, *Parallel Computing* **36**(10-11) (2010) 555-571.
178. M. J. Flynn, Some computer organizations and their effectiveness, *IEEE Transactions on Computation C* **21**(9) (1972) 948-960.
179. W. Gropp, E. Lusk, N. Doss and A. Skjellum, A high-performance, portable implementation of the MPI message passing interface standard, *Parallel Computing* **22**(6) (1996) 789-828.
180. B. Chapman, G. Jost and R. Pas, *Using OpenMP: Portable Shared Memory Parallel Programming (Scientific and Engineering Computation)* (MIT Press, Cambridge, MA, 2007).
181. J. D. Owens, M. Houston, D. Luebke, S. Green, J. E. Stone and J. C. Phillips, GPU Computing, *Proceedings of IEEE* **96**(5) (2008) 879-899.
182. R. M. Calazan, N. Nedjah and L. D. M. Mourelle, Parallel GPU-based implementation of high dimension particle

swarm optimizations, in *Proc. IEEE 4th Latin American Symposium on Circuits and Systems* (2013), pp. 1-4.
183. P. Krömer, V. Snášel, J. Platoš and A. Abraham, Many-threaded implementation of differential evolution for the CUDA platform, in *Proc. 13th Annual Conf. Genetic and Evolutionary Computation* (New York, USA, 2011), pp. 1595-1602.
184. F. Fabris and R.A. Krohling. A co-evolutionary differential evolution algorithm for solving minmax optimization problems implemented on GPU using C-CUDA, *Expert Systems with Applications* **39**(12) (2012) 10324-10333.
185. I. Filali, F. Bongiovanni, F. Huet and F. Baude, A Survey of structured P2P systems for RDF data storage and retrieval, in *Transactions on Large-Scale Data- and Knowledge-Centered Systems III* (Springer, 2011), pp. 20-55
186. M. V. Steen and A. E. Eiben, Peer-to-peer evolutionary algorithms with adaptive autonomous selection, in *Proc. Conf. Genetic and Evolutionary Computation* (2007), pp. 1460-1467.
187. J. L. Laredo, A. E. Eiben, M. Steen, P. A. Castillo, A. M. Mora and J. J. Merelo, P2P evolutionary algorithms: A suitable approach for tackling large instances in hard optimization problems, in *Proc. 14th Int. Euro-Par Conf. Parallel Processing* (2008), pp. 622-631.
188. M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica and M. Zaharia, A view of cloud computing, *Communications of the ACM* **53**(4) (2010) 50-58.
189. Q. Zhang, L. Cheng and R. Boutaba, Cloud computing: State-of-the-art and research challenges, *Journal of Internet Services and Applications* **1**(1) (2010) 7-18.
190. W. P. Lee, Y. T. Hsiao and W. C. Hwang, Designing a parallel evolutionary algorithm for inferring gene networks on the cloud computing environment, *BMC Systems Biology* **8**(5) (2014).
191. K. Meri, M. G. Arenas, A. M. Mora, J. J. Merelo, P. A. Castillo, P. García-Sánchez and J. L. J. Laredo, Cloud-based evolutionary algorithms: An algorithmic study, *Natural Computing* **12**(2) (2013) 135-147.
192. J. A. Parejo, A. Ruiz-Cortes, S. Lozano and P. Fernandez, Metaheuristic optimization frameworks: A survey and benchmarking, *Soft Computing* **16** (2012) 527-561.
193. L. Di Gaspero and A. Schaerf, Easylocal++: An object-oriented framework for flexible design of local search algorithms, *Softw Pract Exp*. **33** (8) (2003) 733–765.
194. S. Luke, et al., Ecj: A java- based evolutionary computation research system (2009), http://cs.gmu. edu/eclab/projects/ecj/
195. S. Cahon, N. Melab and E. G. Talbi, ParadisEO: A framework for the reusable design of parallel and distributed metaheuristics, *Journal of Heuristics* **10**(3) (2010) 357–380.
196. M. Kronfeld, H. Planatscher and A. Zell, The EvA2 optimization framework, in *Proc. Conf. Learning and Intelligent Optimization*, eds. C. Blum, R. Battiti (Springer, 2010), pp 247–250.
197. J. A. Parejo et al., FOM: A framework for metaheuristic optimization, in *Proc. Int. Conf. Computational Science* (Springer, 2003), pp. 886–895.
198. S. Wagner, Heuristic optimization software systems modeling of heuristic optimization algorithms in the heuristic lab software environment, Ph.D. Thesis (Johannes Kepler University, Linz, 2009).
199. S. Ventura et al., JCLEC: A java framework for evolutionary computation, *Soft Computing—A Fusion of Foundations, Methodologies and Applications* **12**(4) (2008) 381–392.
200. E. Alba et al., MALLBA: A software library to design efficient optimisation algorithms, *International Journal of Innovative Computing and Applications* **1** (2007) 74–85.
201. J. Brownlee, OAT: The optimization algorithm toolkit, Technical Report (Complex Intelligent Systems Laboratory, Swinburne University of Technology, 2007).
202. M. Lukasiewycz, F. R. M. Glass and S. Helwig, Opt4j — The optimization framework for java (2009), http://www.opt4j.org
203. S. Ye, G. Dai, L. Peng and M. Wang, A hybrid adaptive coevolutionary differential evolution algorithm for large-scale optimization, in *Proc. IEEE Congress on Evolutionary Computation* (Bejing, 2014), pp. 1277-1284.
204. X. Li and X. Yao, Cooperatively coevolving particle swarms for large scale optimization, *IEEE Transactions on Evolutionary Computation* **16**(2) (2012) 210-224.
205. M. N. Omidvar, X. Li, Y. Mei and X. Yao, Cooperative co-evolution with differential grouping for large scale optimization, *IEEE Transactions on Evolutionary Computation* **18**(3) (2013) 378-393.
206. M. Lozano, D. Molina and F. Herrera, Editorial scalability of evolutionary algorithms and other metaheuristics for large-scale continuous optimization problems, *Soft Computing* **15** (11) (2011) 2085-2087.
207. K. Tang, X. Li, P. N. Suganthan, Z. Yang and T. Weise, Benchmark functions for the CEC'2010 special session and competition on large scale global optimization, Technical Report (Nature Inspired Computation and Applications Laboratory, USTC, China, 2009).
208. X. Li, K. Tang, M. Omidvar, Z. Yang and K. Qin, Benchmark functions for the CEC'2013 special session and competition on large scale global optimization, Technical Report (Evolutionary Computation and Machine Learning Group, RMIT University, Australia, 2013).
209. D. Molina, M. Lozano and F. Herrera, MA-SW-Chains: Memetic algorithm based on local search chains for large scale continuous global optimization, in *Proc. IEEE Congress on Evolutionary Computation* (2010), pp 1-8.
210. A. LaTorre, S. Muelas and J.-M. Pena, Multiple offspring sampling in large scale global optimization, in *Proc. IEEE Congress on Evolutionary Computation* (2012), pp 1-8.
211. Z. Yang, K. Tang and X. Yao. Large scale evolutionary optimization using cooperative coevolution, *Information Sciences* **178** (2008) 2985-2999.
212. A. E. I. Brownlee, J. A. W. McCall and Q. Zhang, Fitness modeling with Markov networks, *IEEE Transactions on Evolutionary Computation* **17**(6) (2013) 862-879.
213. Y. Jin, Surrogate-assisted evolutionary computation: recent advances and future challenges, *Swarm Evolutionary Computation* **1**(2) (2011) 61-70.

214. Y. Tenne, K. Izui and S. Nishiwaki, Dimensionality-reduction frameworks for computationally expensive problems, in *Proc. IEEE Congress on Evolutionary Computation* (2010), pp. 1-8.
215. B. Liu, Q. Zhang and G. G. E. Gielen, A Gaussian process surrogate model assisted evolutionary algorithm for medium scale expensive optimization problems, *IEEE Transactions on Evolutionary Computation* **18**(2) (2014) 180-192.
216. D. Lim, Y. Jin, Y.-S. Ong and B. Sendhoff, Generalizing surrogate-assisted evolutionary computation, *IEEE Transactions on Evolutionary Computation* **14**(3) (2010) 329-355.
217. A. E. Eiben and S. K. Smit, Parameter tuning for configuring and analyzing evolutionary algorithms, *Swarm and Evolutionary Computation* **1** (2011) 19-31.
218. M. Birattari, Z. Yuan, P. Balaprakash and T. Stutzle, F-race and iterated F-race: An overview, in *Experimental Methods for the Analysis of Optimization Algorithms* (Springer, 2010), pp. 311-336.
219. M. Lopez-Ibanez, J. Dubois-Lacoste, T. Stuzle and M. Birattari, The irace package, iterated race for automatic algorithm configuration, Technical Report (IRIDIA, Universite Libre de Bruxelles, 2011).
220. T. Bartz-Beielstein, C. Lasarczyk and M. Preuss, The sequential parameter optimization toolbox, in *Experimental Methods for the Analysis of Optimization Algorithms* (Springer, 2010), pp. 337-360.
221. F. Hutter, H. H. Hoos, K. Leyton-Brown and T. Stutzle, ParamILS: An automatic algorithm configuration framework, *Journal of Artificial Intelligence Research* **36** (2009) 267-306.
222. F. Hutter, H. H. Hoos and K. Leyton-Brown, Sequential model-based optimization for general algorithm configuration, in *Proc. 5th Int. Conf. Learning and Intelligent Optimization* (Springer, 2011), pp. 507-523.
223. J. Santamaria, S. Damas, O. Cordon and A. Escamez, Self-adaptive evolution toward new parameter free image registration methods, *IEEE Transactions on Evolutionary Computation* **17**(4) (2013) 545-557.
224. B. B. Barrios, Q. Castella, A. A. Juan, H. R. Lourenco and M. Mateo, ILS-ESP: An efficient, simple, and parameter-free algorithm for solving the permutation flow-shop problem (Barcelona GSE Working Paper Series, 2012).
225. G. Caldas, R. Schirru, FPBIL: A parameter-free evolutionary algorithm, in W. Kosinski (Eds.) *Advances in Evolutionary Algorithms* (I-Tech Education and Publishing, 2008), pp. 49-75.
226. T. *Erl*, Z. Mahmood and R. Puttini, *Cloud Computing, Technology & Architecture* (Prentice Hall, 2013).
227. J. Chen, B. Xin, Z. Peng, L. Dou and J. Zhang. Optimal contraction theorem for exploration-exploitation tradeoff in search and optimization, *IEEE Transactions on Systems, Man, and Cybernetics Part A:Systems and Humans*, **39**(3) (2009) 680-691.
228. P.A.N. Bosman and D. Thierens, The balance between proximity and diversity in multiobjective evolutionary algorithms, IEEE Transactions on Evolutionary Computation **7**(2) (2003) 174-188.
229. M. Črepinšek, S.-H. Liu, and M. Mernik. Exploration and exploitation in evolutionary algorithms: A survey. ACM Comput. Surv. **45**(3), (2013), 33 pages.
230. B. Ostadmohammadi Arani, P. Mirzabeygi and M. Shariat Panahi. An improved PSO algorithm with a territorial diversity-preserving scheme and enhanced exploration-exploitation balance, *Swarm and Evolutionary Computation*, **11** (2013) 1-15.
231. A. LaTorre, S. Muelas and J.M. Peña, A MOS-based dynamic memetic differential evolution algorithm for continuous optimization: a scalability test, *Soft Computing*, **15**(11) (2011) 2187-2199.
232. J.E. Fieldsend, R.M. Everson and S. Singh, Using unconstrained elite archives for multiobjective optimization, *IEEE Transactions on Evolutionary Computation*, **7**(3) (2003), 305-323.
233. M.G. Epistropakis, X. Li and E.K. Burke, A dynamic archive niching differential evolution algorithm for multimodal optimization, in Proc. IEEE Congress on Evolutionary Computation (2013) pp 79-86.
234. C.K. Chow and S.Y. Yuen, An Evolutionary Algorithm That Makes Decision Based on the Entire Previous Search History, *IEEE Transactions on Evolutionary Computation* **15**(6) 741-769.
235. K. Sörensen. Metaheuristics—the metaphor exposed. International Transactions in Operational Research, **22**(1) (2015) 3-18.
236. A. Piotrowski, J. Napiorkowski, P.M. Rowinski, How novel is the "novel" black hole optimization approach? *Information Sciences*, 267 (2014) 191-200.
237. L. Bianchi, M. Dorigo, L.M. Gambardella, W.J. Gutjar, A survey on metaheuristics for stochastic combinatorial optimization, *Natural Computing*, **8**(2) (2009), 239-287.
238. K. Narukawa, T. Rodemann, Examining the Performance of Evolutionary Many-Objective Optimization Algorithms on a Real-World Application, in Proc of *Sixth International Conference on Genetic and Evolutionary Computing (ICGEC, 2012)*, pp. 316-319.
239. R.W. Garden and A.P. Engelbrecht, Analysis and Classification of Optimisation Benchmark Functions and Benchmark Suites, in *Proc. IEEE Congress on Evolutionary Computation* (Bejing, 2014), pp. 1641-1649.
240. D.H. Wolpert and W.G. Macready, No free lunch theorems for optimization, *IEEE Transactions on Evolutionary Computation*, **1**(1) (1997) 67-82.
241. C. García-Martínez, F.J. Rodríguez and M. Lozano, Arbitrary function optimisation with metaheuristics: No free lunch and real-world problems: No free lunch and real-world problems, *Soft Computing*, **16**(12) (2012) 2115-2133.
242. F. Neri and C. Cotta, Memetic algorithms and memetic computing optimization: A literature review, *Swarm and Evolutionary Computation* **2** (2012) 1-14.
243. S. Das, M.H. Lim, Guest editorial: Special issue on engineering applications of memetic computing, *IEEE Transactions on Systems, Man and Cybernetics Part C: Applications and Reviews,* **42**(5) (2012) 609-611.
244. X.S. Chen, Y.S. Ong, M.H. Lim and K.C. Tan, A Multi-Facet Survey on Memetic Computation, *IEEE Transactions on Evolutionary Computation* **15**(5) (2011) 591-607.