Operations Research Forum (2021) 2: 36
https://doi.org/10.1007/s43069-021-00068-x

EXPOSITORY REVIEWS

Review on Nature-Inspired Algorithms

Wael Korani[1] · **Malek Mouhoub**[1]

Received: 5 May 2020 / Accepted: 17 May 2021 / Published online: 16 July 2021

Abstract
Optimization and its related solving methods are becoming increasingly important in most academic and industrial fields. The goal of the optimization process is to make a system or a design as effective and functional as possible. This is achieved by optimizing a set of objectives while meeting the system requirements. Optimization techniques are classified into exact and approximate algorithms. Nature-inspired (NI) methods, a sub-class of approximate techniques, are widely recognized for providing efficient approaches for solving a wide variety of real-world optimization problems. In this paper, we discuss many scenarios where we can or cannot use different NI methods in tackling real-world optimization problems. We also enrich our survey with many studies for the reader to prove the efficiency and efficacy of using NI methods to tackle many real-world applications. Therefore, NI methods should be considered as alternative reliable approaches in the absence of exact methods to provide satisfactory solutions.

Keywords Optimization · Nature-inspired algorithms · Swarm Intelligence · Evolutionary Computation · Genetic Algorithms

1 Introduction

Constraint optimization is a search process for finding the best solution(s) meeting a set of problem requirements (constraints) and optimizing one or more objectives [1]. The main challenge is to find the optimal solution (assuming one single objective function) in a reasonable amount of time. Optimization algorithms are classified into: exact and approximate. While they guarantee to return the optimal solution, exact algorithms are limited, in practice, by their exponential time cost. For those problems

✉ Malek Mouhoub
mouhoubm@uregina.ca

Wael Korani
wmk182@uregina.ca

[1] Department of Computer Science, University of Regina, 3737 Wascana Parkway, SK S4S0A2 Regina, Canada

where these methods fail to return an optimal solution within a reasonable time, approximate methods including metaheuristics can be a good alternative. Indeed, metaheuristics do not explore the entire search space, and therefore trade the quality of the solution returned for the running time. Metaheuristics include nature-inspired (NI) which mimic a natural phenomenon from biology, physics, or ethology [2].

In the last decades, there has been a lot of attention devoted to NI algorithms, which are classified into: single-based and population-based solutions. In 1960s, L.J Fogel et al. introduced the basic concept of Evolutionary Programming (EP) [3]. In 1964, Rechenberg et al. introduced the evolutionary strategies (ESs). In 1975, John Holland introduced the basic concept of genetic algorithms (GAs) [4]. In 1983, Kirkpatrick et al. proposed the simulated annealing (SA) algorithm that is inspired by the simulated annealing phenomena [5]. In 1995, Kennedy and Eberhart introduced the particle swarm optimization (PSO) algorithm that mimics the movement of a flock of birds [6]. In 2002, Passino introduced the bacterial foraging (BF) algorithm that is built on the foraging behavior of *E. coli* bacteria [7]. In 2007, Atashpaz-Gargari introduced the first idea of imperialist competitive algorithm (ICA) that is built on the imperialist competitive idea [8]. In 2013, Cheng et al. introduced the competitive swarm optimizer (CSO) that is built on the same concept of PSO with some diffirences [9]. There are many other NI algorithms that have been proposed and achieved promising results [10]. In the paper, we review just few algorithms that have been implemented in too many different applications and proved their efficacy.

The focus of our paper is on when we can use NI algorithms, and what the main drawbacks of using them. The remaining of this paper is structured as follows. Section 2 presents an overview of optimization algorithms, including exact and approximation methods. Section 3 describes the basic concept of single-solution based NI algorithms, such as SA. Section 4 presents the population-based NI algorithms. Section 4.1 outlines the basic steps of Evolutionary Computation (EC) techniques: GAs, ESs, and EP are discussed as examples of EC algorithms. Section 4.2 presents a special class of NI algorithms called "Swarm Intelligence" (SI). The last sections describe the methodology to assess, in practice, the performance of a NI technique.

2 Overview of Optimization Algorithms

In [11], Archetti et al. classified global optimization algorithms into two main categories: exact (deterministic) and approximate (probabilistic), as shown in the taxonomy presented in Fig. 1. In addition, Fister et al. classified many swarm intelligence and bio-inspired algorithms in a well-organized way [12]. A taxonomy of data-driven metaheuristics has also been reported by El Ghazali in [13].

2.1 Exact and Approximate Algorithms

Exact methods are systematic search techniques that explore the entire search space in order to find the optimal solution, such as dynamic programming, backtracking and its variants, branch-and-bound, and constraint programming techniques [14].

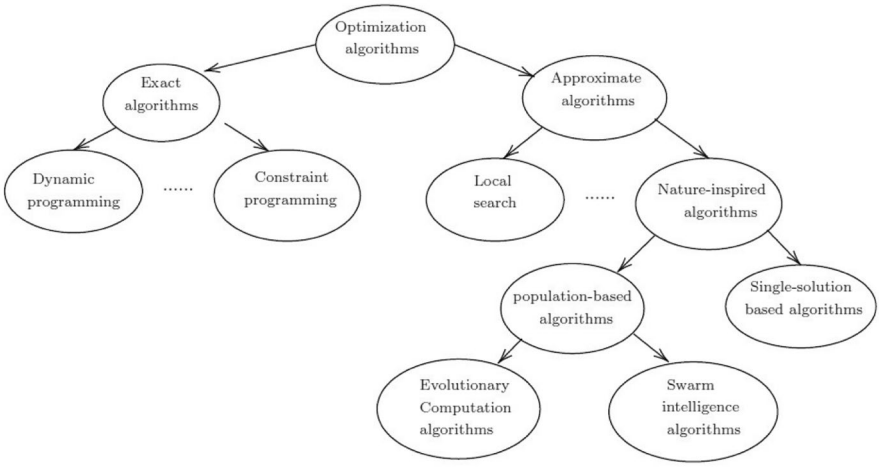

Fig. 1 Optimization algorithms classification

These algorithms are deterministic, meaning that they follow the same search path across different runs for solving the same problem instance. While they guarantee to return the optimal solution, in some cases, these methods can be unpractical due to their exponential time cost, especially when dealing with NP-hard problems [15]. Exact algorithms can be used to solve hard optimization problems that have different levels of difficulties, such as size of the problem, hardness (tightness) of the constraints, and nonlinear characteristics. For example, in [16], the traveling salesman problem has been successfully solved for different instances using an exact method. In other words, some large instances might be solved by exact methods in a reasonable time frame, while smaller instances fail to be solved. For many NP-hard problems, there is a region, called the phase transition, that includes the hardest problems to solve, regardless of the problem size. The phase transition can be seen as the borderline between solvable and unsolvable problems [17].

Approximate algorithms are built upon the concept of randomness guided by different rules, without a guarantee of convergence. NI algorithms represent well-known and efficient subclass of approximate algorithms that is inspired by different natural phenomena. NI are considered general-purpose algorithms, and they are suitable for a wide range of problems and problem instances. This class of algorithms have one (single-solution-based algorithms) or more (population-based algorithms) candidate solutions (also called agents). These latter are evaluated and rewarded with fitness values. Candidate solutions change over time depending on the application of operators, following some random parameters. NI techniques are designed following a trade-off between two main strategies: exploration and exploitation [10]. Exploration, also known as diversification, is the process of exploring a wider search space. Exploitation, also known as intensification, is the process of exploiting the best solutions found and intensifying the search locally. Using a good balance between these two strategies will help delivering satisfactory solutions in a reasonable time frame for black-box problems (problems that do not have clear

properties) and large-size industrial problem instances. NI have been used in many different industrial applications and has proven both efficiency and efficacy [18, 19]. In addition, this class of algorithms is the corner stone of many machine learning techniques [13]. The main disadvantage is that inconsistency cannot be proven for those overconstrained problems; however, it has the ability to efficiently produce good solutions.

2.1.1 Why and When to Use NI Algorithms

The choice of an appropriate algorithm for a given optimization problem is an open question in the optimization research community. There is no claim that a specific optimization problem can be solved with only one optimization algorithm, but the selection process of the right algorithm to solve a given problem in a reasonable time frame using minimal resources is a difficult and a tedious task. For instance, a discontinuous objective function cannot be effectively solved using a classical gradient-based approach, such as hill-climbing, while the highly nonlinear problems can be solved with NI population-based methods [20]. Similarly, a constraint problem can be solved in polynomial time if its constraint graph representation is a tree [14].

There are several factors should be taken in consideration: the size of the problem (small, medium, and large), the type or structure of the problem (linear or nonlinear, continuous or discrete), the time limit [10] (fast or slow), the desired quality of solutions (exact or approximate), the availability of resources, the easiness of implementing the algorithm [20]. The structure of an optimization problem plays a significant role over the size of the problem, because some easy–medium and large-size problems could be solved by exact algorithms in reasonable time frame. In some cases, a user needs an efficient algorithm to reduce the number of function evaluations (time), because each iteration may take few hours or even weeks as in case of optimizing the mixing index of biomass optimization problem.

The first step to solving an optimization problem is to analyze its time and space complexity, and determine whether the problem can be reduced to a tractable one from the literature. In other words, a user might search in literature for the same or similar problem to the one that the user wants to tackle. The next step is to find the best appropriate optimization algorithm to solve that given problem in a time frame fit the user requirements. User should be careful and follow the recommended tuning parameters from literature to avoid a deteriorated performance of the selected algorithm. Otherwise, a related problem could be found, so that its methodology might be followed. We also want to highlight those portfolio-based algorithm selectors that can be used as guidance for the appropriate choice of the nature-inspired technique depending on the problem instance features and properties [21].

Exact algorithms are the best choice whenever they can be used to solve or assist in solving a given problem within a reasonable time frame, such as some polynomial time problems and some large-scale linear continuous problems. Therefore, it will be an unwise decision to use NI algorithms to solve easy optimization problems, where an exact algorithm is available.

On the other hand, the use of *NI algorithms* is desirable and can be advantageous for approximate solutions. Using NI algorithms can be the best choice when a simplified model is used or the model parameters are estimated using inexact or limited data [22]. For such problems that are inaccurately represented (due to incomplete/uncertain information about constraints or objectives), using approximate methods can be better than time-consuming exact methods. In addition, NI algorithms could be used when a reliable exact method is unavailable or the available exact method is computationally undesirable (excessive time and or space). In those cases, NI algorithms can sometimes produce a reasonably good solution with minimal time and space requirements. NI algorithms can also be used to enhance the performance of an existing exact method by providing good starting solutions or guiding the search. Furthermore, NI algorithms have been recommended to be used to save time when the same problem is going to be solved frequently, because they are simple to implement and easy to understand. Moreover, they help to gain insight into complex problems, while using minimal resources [23].

3 Single-Solution-Based NI Algorithms

Single-solution-based (SS-based) NI class of algorithms is also known as trajectory methods. Any member of this class starts with a single agent, candidate solution, and improves its fitness value in each generation (iteration). The candidate solution moves through neighborhoods or search trajectories within a predefined search space [24]. The trajectories are produced by an iterative mechanism and move from one position to another in the predefined search space. This class performs two fundamental operations into two phases: generation and replacement. In the following subsection, we discuss the SA algorithm, a popular and well-known single-solution-based NI algorithm.

3.1 Simulated Annealing Algorithm

Annealing is a physical process where a certain alloy of metal, glass, or crystal is heated above its melting point. Then, it is cooled until it is eventually frozen into a perfect crystalline structure. The annealing process can produce high-quality materials. The SA algorithm is a single-solution-based algorithm, and it is also known as Boltzmann annealing. It updates its candidate solution by random ascent that moves to avoid being trapped in a local minimum. It is considered a Monte Carlo algorithm for finding a global minimum for continuous functions. According to Boltzmann distribution, the probability of a physical system (P_α) being at state α with energy E_α at temperature T is given by:

$$P_\alpha = \frac{1}{Z} e^{\left(\frac{-E_\alpha}{K_B} T\right)}, \tag{1}$$

where K_B is the Boltzmann constant, T is the absolute temperature, and Z is a partial function described by:

$$Z = \sum_{\beta} e^{(\frac{-E_\beta}{K_B}T)}, \tag{2}$$

where the summation is over all states β with energy E_β at temperature T. The Boltzmann distribution exhibits uniform preference for all states at high T regardless of the energy. However, when T decreases to zero, only states with minimum energy have nonzero probability of occurrence.

Several studies have been conducted to enhance the performance of the SA search algorithm in terms of accuracy and convergence rate. In [25], Harold et al. introduced a variant of SA that replaces the Boltzmann probability density with the Cauchy probability density. In [26], the generalized SA was proposed to generalize both Cauchy annealing [25] and Boltzmann annealing [5]. Microcanonic annealing (MA) is a variant of SA, that is built on the Creutz algorithm rather than the Metropolis algorithm [27]. In [28], Dueck et al. introduced a variant of SA known as threshold accepting (TA) methods. In [28], the results showed that TA outperforms SA in terms of search time.

3.2 Advantages, Limitations, and Applications of SS-based Algorithms

In general, SS-based methods are effective, because they provide sufficient knowledge about their behavior. In case of large instances, SS-based algorithms might perform better than population-based (P-based) algorithms [29]. SS-based algorithms achieved a competitive performance against P-based algorithms on different Traveling Thief Problem (TTP) instances [29]. SA is more suitable for the job shop scheduling problem than other algorithms, such as a tailored heuristic algorithm [30]. SA is the most dominant SS-based NI algorithm that has been used extensively in recent decades. The main advantage of SA is that it is a simple search algorithm to implement, and it is appropriate for solving black-box optimization problems [31, 32]. In addition, SA has a fast convergence compared to exact methods and other heuristic algorithms, and it is suitable to be used with problems that have a large number of local minima [33]. In [34], SA achieved better results than Bayesian algorithms (they are variations on a method developed by H. J. Kushner) in terms of computation time and number of function evaluations in optimizing the results of a computer simulation. In [30], the results showed that SA produced better results than tailored heuristic algorithm in solving the job shop scheduling problem. SA has been used in optimal design problems where many researchers consider SA as a tool in the development process of optimal experimental design [35–38].

There are some *limitations* of using SS-based algorithms. They suffer from many function evaluations to reach the global optimum, which increase in search time [32]. The initial temperature of the cooling schedule should be set to an appropriate high value, which is another issue. A low initial temperature may cause the algorithm to get stuck in a local minimum, and a high initial temperature may cause difficulty in reaching the global optimal solution. In addition, the number of iterations at each temperature step should be large enough to exploit each region in the search space [32]. Furthermore, the cooling schedule should be carefully selected, because it may affect

the quality of solutions. Users who have limited knowledge about SA may find the selection process of these parameters to be difficult. SA is not suitable for problems that have a limited number of local minima [33]. In SA, the neighborhood operators cannot effectively deal with clustered data when solving the vehicle routing problem. Thus, it is recommended to combine the improved SA with a data clustering method, such as k-means clustering [39]. In [40], it was shown that SA does not guarantee to solve a large instance of graph coloring problem. The authors suggested to split the time interval into several runs and return the minimum value over all runs.

SA was proposed to solve combinatorial optimization *applications*, and it has been successful in this regard. For instance, in [41], Emden et al. applied the SA algorithm to solve the airline crew pairing problem. In [42], Rosmalina et al. implemented SA along with a heuristic method to solve the railway crew scheduling problem. In [177], Wong et al. successfully implemented SA to solve the layout-routing of electronic circuits problem. In [170], Supatcha et al. implemented the SA algorithm along with a hill-climbing local search method to solve large-scale aircraft trajectory planning. In [39], an improved SA variant was applied to solve the vehicle routing problem with time windows. In [40], Alper et al. applied the SA algorithm to solve the graph coloring problem. In [31, 43], Delahaye et al. implemented the SA algorithm to solve two NP-hard combinatorial optimization problems: the traveling salesman problem and the knapsack problem. SA has been implemented as well to solve continuous optimization problems. In [44], the enhanced simulated annealing (ESA) variant was proposed to solve multimodal functions. In [45], David et al. have adapted SA for solving the quadratic assignment problem. In [46], Alfonsas introduced an adapted version of SA called M-SA-QAP that has an advanced formula for calculating the initial and final temperature, and he proposed an original cooling schedule with oscillation that allows for both decreasing and increasing the temperature.

4 Population-Based NI Algorithms

Population-based NI algorithms use a group of solutions rather than one single solution. This class of algorithms has two main subcategories: EC and SI. In EC algorithms, individuals in the population are updated through recombination and mutation operators. These algorithms are built on Darwin's evolutionary theory. In SI algorithms, individuals in the population communicate in an intelligent way to explore different regions in the search space rather than using individual cognitive abilities alone. These algorithms are built on collective behavior of an organized group of, e.g., insects, animals, or plants. P-based NI algorithms include many well-known algorithms, such as GAs, ESs, EP, PSO, and BF.

4.1 Evolutionary Computation Algorithms

EC algorithms are also known as evolutionary algorithms (EAs). EC algorithms are built on Darwinian principle of nature's capability to evolve and adapt to their environment. This class of algorithms includes genetic algorithms (GAs), evolutionary

strategies (ESs), evolutionary programming (EP), genetic programming (GP), and so forth. All members of this class have the same idea of simulating the evolution of the candidate solutions using some operators: selection, recombination, mutation, and reproduction. These algorithms have been successfully implemented on different optimization problems, such as data mining and knowledge discovery [47].

4.1.1 Genetic Algorithms

GAs take the basic idea of genetics in order to artificially construct an optimization search algorithm that is robust and can tackle complex black-box problems [48]. In 1975, John Holland introduced the basic concept of GAs [4]. The basic mechanism of GAs involves nothing more than random number generations, string copies, and partial string exchanges [49]. GAs have three basic operators: initialization, selection, and reproduction. In GAs, the initial population of chromosomes (called candidate solutions) is *generated randomly* in the problem search space and then *encoded* (as binary or real value). The solution process includes many generations (iterations); and each generation consists of many function evaluations (objective function evaluations) of candidate solutions. The size of the initial population is a controversial question in the literature. One group of researchers suggested using a population that is sufficiently large to enhance the search diversity [49]. However, another group of researchers introduced the term micro-genetic (microGA) that refers to a small initial population with reinitialization [50]. In [51], Krishnakumar conducted the first comparison between the microGA and the basic GA. He concluded that microGA is faster and provides better results when applied to two stationary functions and a real-world engineering problem (a wind-shear controller task) [51].

4.1.1.1 Genetic Algorithms Variants GAs are adaptable algorithms, so that they have many *variants* in different applications. A real-coded GA (the encoding is over the real number) can be used for global continuous optimization problems. The *crossover operator* is the main search operator in the GA. Researchers proposed many crossover operators for the real-coded GAs. In [52], Syswerda introduced the concept of the Uniform Crossover (UX) operator. In [53], Ono et al. introduced a real-coded GA using Unimodal Normal Distribution Crossover (UNDX). In [54], Ono et al. introduced a crossover approach that combines UNDX and UX. The results showed that the proposed crossover approach can solve more different functions than GA using only the UNDX. In [55], Deb et al. introduced a crossover operator called Simulated Binary Crossover (SBX). In [56], Sánchez et al. introduced a hybrid crossover operator that generates multiple descendants from two parents, and the best two offspings will replace the parents in the next generation. In [57], Eshelman et al. introduced the blend crossover (BLX-α), but it faces some difficulties when it is used to solve non-separable functions. In [58], Takahashi et al. introduced a crossover operator that combines the BLX-α and the Independent Component Analysis (ICA). *Mutation operator* has been used to improve the performance of GAs for functions optimization. In [59], Munteanu et al. introduced a mutation operator for real-coded GA called principal component analysis mutation (PCA-mutation). In [60], Korejo et al. proposed the Directed Mutation (DM) operator that allows a GA to explore promising

areas in the search space. In [61], the authors introduced a popular non-domination-based genetic algorithm for multi-objective optimization.

4.1.2 Evolution Strategies

ESs are a class of EA introduced in 1964 by Rechenberg and Schewefel [62]. The first ES algorithm, (1+1)-ES (two membered ES), is a simple mutation-selection schema, and it has a population of two individuals. The two membered ES algorithm produces a single offspring using normal, Gaussian, distribution mutation. For the next generation, the best individual is elected by a selection operator. Rechenberg developed later the concept of the multimembered ES, (μ+1)-ES, where μ is denoted to the number of parents, and they collaborate to generate λ offsprings. In the multimembered ES, the best individuals among offsprings will be the parents of the next generation, while the current parents will be removed. In [63], Schwefel introduced two further variants of multimembered ES: (μ+λ)-ES and (μ, λ)-ES. In the (μ+λ)-ES, parents (μ) generate offsprings (λ) using recombination and mutation operators. The selection operator then selects the best individuals, equal to the number of parents, among the parents and offsprings (μ+λ) and discards the rest. In (μ, λ)-ES, the number of generated offsprings is greater than the number of parents. Then, the best individuals, equal to the number of parents, is selected from the offsprings (λ), and the parents of the offsprings are discarded no matter how good or bad their fitness value. In most recent variants of ES, the population of size μ is used, and an additional operator called recombination (ρ) is implemented. There are two other variants were built on the concept of that recombination operator: (μ/ρ+λ)-ES and (μ/ρ, λ)-ES

4.1.3 Evolutionary Programming (EP)

EP is a stochastic optimization search approach that belongs to the EAs family. In [3], in 1960s, L.J Fogel et al. introduced the basic idea of EP to artificial intelligence. In [64], in 1990s, David B Fogel introduced again the EP concept to solve many problems, such as numerical and combinatorial optimization and machine learning [65–67]. Mutation is the main operator in EP, while crossover operator is the main operator in GAs. The EP algorithm has two major operators: mutation and selection, but it does not have any kind of recombination operators. Mutation generates offsprings, while selection selects the best individuals among parents and offsprings for the next generation. In EP, the initial population is randomly generated based on a density function, and each individual is evaluated using an objective function. Then, mutation is implemented for generating new offsprings. The mutation operator is implemented by perturbing each parent in the population . This is done by adding a random number of specific distribution, e.g., normal distribution. In [173], David B Fogel et al. introduced the meta-evolutionary programming (meta-EP) idea. Meta-EP has the capability to discover the appropriate degree of perturbation for a given problem that makes the meta-EP has a self-adaptation of variances for the mutation operator. The self-adaptive EP is widely used for continuous parameter optimization problems. The first non-Gaussian mutation was introduced

in the mid-1990s by Yao. In [68], Yao et al. introduced an EP variant called fast EP (FEP), which uses Cauchy instead of Gaussian mutation operator as the primary search operator. Cauchy probability distribution has a much longer tail; therefore, the offsprings could be totally different from their parents. In [69], Yao et al. later introduced an improved FEP (IFEP) variant. In [70], Lee et al. introduced a generalized version of FEP by using Lévy probability distribution, which is a general case of Cauchy probability distribution.

4.1.4 Advantages, Limitations, and Applications of EC Algorithms

EC algorithms receive much attentions for their *advantages* and capability in solving hard real-world optimization problems in different fields of science. They have been applied to problems that could not be solved using heuristic algorithms. In [71], David B Fogel summarized the main advantages of EC algorithms. The main advantage is that they are conceptually simple, and they can be applied to any optimization problem. EC algorithms can be applied using any representation; therefore, the same procedure can be used for discrete and continuous optimization problems. In addition, EC algorithms outperform classical methods on real-world problems, and they have significant advantage over classical methods in solving mutlimodal functions. Furthermore, EC algorithms provide a methodological framework that is usable as it is or can be combined with other optimization method. They can be used as appropriate methods when problems have: dynamic situations, i.e., the goal or constraint changes over time [72], parameter adjustments, rough or discontinuous landscape, and disturbed fitness measurements [73]. EC algorithms are a highly parallel process, and they have the capability to optimize their parameters as a part of the search for optimal solutions. Perhaps the greatest advantage is that EC algorithms have the ability to address problem where human experts do not exist. The EC community has been criticized for considering uniform standard instances, which are much easier than real-world applications. However, in [74], Dimopoulos et al. introduced a significant work to present the contribution of EC algorithms towards realistic problems taken from manufacturing plants. In theory, the effectiveness of an EC algorithm depends on the relationship between crossover and mutation as applied to a chosen representation [71]. They can be combined with other classical or heuristic algorithms [32]. The GA achieves much better solution than *exact approaches* in solving pipe optimization problem in terms of speed and quality of solutions [75]. In [76], Chu et al. introduced a GA to solve the generalized assignment problem. In [77], Alba et al. introduced a good survey to prove that parallel GAs enhances the computation over regular GAs, and helps in producing better solution. In [78], the authors implemented adaptive genetic algorithm along with fuzzy logic to propose a classifier to diagnosing heart disease. In [79], the authors implemented the genetic algorithm to solve supercapacitor charging problem.

However, EC algorithms have some *limitations* as well. EC algorithms should not be used to solve a given problem where there is a traditional method that can solve it, because EC algorithms cannot be better with less computational effort [73]. In [80], Back concluded that EC algorithms are not appropriate methods to solve strongly

convex problems. In [81], Moslemipour et al. summarized some drawbacks about GAs. They may find a suboptimal solution, and they are not guaranteed to reach the global solutions. The mutation and crossover rates affect the stability of the algorithm, and they have difficult encoding schema. In [82], Leung et al. concluded that GAs suffer from premature convergence. The authors suggested that increasing the population size plays an important role to help overcome this problem. The effectiveness of GAs depends on the population size and crossover and mutation rates; therefore, these parameters should be selected carefully [33]. For instance, increasing the population size or the number of generation will lead to an increase in search time. In addition, the formulation of the fitness function is not an easy process. In [83], a binary-coded GA presents unsatisfactory results in solving multimodal functions with respect to real-coded differential evolution (DE). DE may offer easier convergence by increasing the number of parents and reducing the scaling factor; however, DE, may then suffer from high computational time. The basic EP algorithm has a slow conversion in solving some multimodal optimization problems [69]. However, in [84], EP outperforms GAs in solving constraint optimization problems. ESs and EP share common features, but EP does not have a recombination operator. ES algorithms were developed to solve continuous optimization problems. They have the flexibility to develop a new robust method for a given problem [73]. In [85], Swayamsiddha et al. concluded that the differential evolution algorithm provided the best results for solving a nonlinear system identification compared to GA and PSO. The differential evolution algorithm outperforms GA in solving a suit of benchmarks in terms of number of function evaluation [86]. In [87], the Bayesian optimization algorithm achieved better performance than a complex multi-population GA in tackling Nurse Scheduling Problem (NSP).

EC algorithms have been extensively used in many *applications*. In routing problems, the traveling salesman problem [88] is one of the most well-known combinatorial optimization problems that has been solved using EC algorithms [89]. In scheduling problems, the job shop scheduling problem is an NP-complete problem that has been solved using EC algorithms [90]. In packing problems, a design of layout for integrated circuits is a well-known example [91]. EC algorithms have been implemented in different design applications, such as finite impulse response (FIR) [92, 93], infinite impulse response (IIR) [94, 95], signal processing [96, 172], integrated circuit design [96, 176, 180], artificial neural networks [175, 178, 179], telecommunication [168, 171], and engineering applications [96, 97]. GAs have been implemented in different applications in mechanical engineering: material science and manufacturing [98]. EC algorithms have been used in system identification and simulation applications [99]. System identification is used for model structure selection and parameter estimation, while system simulation process is to determine how the system will behave. In [100], Abd Samad introduced a good survey about the usage of EC algorithms in the field of system identification in the last 40 years. In addition, EC algorithms have been implemented for critical applications: on-line and off-line control system engineers [101]. In [102], Fleming et al. introduced a survey of using EC algorithms in control system, and they concluded the importance of EC algorithms in control system applications. Furthermore, EC algorithms have been used extensively in data mining [103], image processing [104, 105], and classification [106].

4.2 Swarm Intelligence Algorithms

SI algorithms are inspired by the collective behavior of species, such as fish, birds, ants, wasps, bees, termites, and bacteria [107]. The SI theory is built on the social behavior of those species that compete to obtain the best source of food. The SI Population consists of particles (agents) that cooperate by an indirect communication medium to improve their fitness in the search space. PSO is the most dominant SI algorithm, and there are thousands of papers about its variants and applications in different fields. We give few popular examples of SI algorithms: PSO, BF, and ICA.

4.2.1 The Particle Swarm Optimization Algorithm

In 1995, Eberhart and Kennedy introduced the first idea of particle swarm optimization [6, 108, 109]. PSO mimics the movement of a flock of birds. Each bird in the flock is associated to a particle (candidate solution). The position of each particle in the search space is updated based on the previous best position of the particle itself (local position) and the best position of the entire flock (global position). The PSO algorithm updates the position of each particle using the following equation [108]:

$$x_{id}^{k+1} = x_{id}^{k} + v_{id}^{k+1}, \tag{3}$$

where x_{id} is the position of a particle i, the superscript k denotes the iteration rank, and v_{id} is the velocity of the particle i. The velocity of the particle i is updated using the following equation:

$$v_{id}^{k+1} = v_{id}^{k} + c_1 \times r_1[P_{id}^{k} - x_{id}^{k}] + c_2 \times r_2[P_{gd}^{k} - x_{id}^{k}], \tag{4}$$

where the v_{id}^{k} is the previous velocity of the particle i that provides the necessary momentum for moving around the search space. The constants c_1 and c_2 are also known as the acceleration coefficients, and r_1 and r_2 are uniform distribution random numbers in range [0,1]. P_{id}^{k} is the local best position for the particle i at iteration k, and P_{gd}^{k} is the global best position at iteration k.

PSO has too many variants that have been introduced and marked the history of PSO over the last three decades. In [109, 110], Shi and Eberhart proposed a variant of PSO called inertia weight PSO that has an extra parameter called inertia weight (ω). In [110], Shi and Eberhart proposed a variant called PSO time varying inertia weight (PSO-TVIW) in which the inertia weight decreases along with time. In [111], Zheng and Ma proposed another PSO variant that increases the inertia weight value during the course of the run. In [109], Shi et al. reported that large inertia weight values enhance the global search, while small inertia weight values enhance the local search. In [112], Clerc and Kennedy introduced the canonical PSO variant, which has a constriction factor that helps in controlling the convergence properties of the particles. In [113], Mendes and Kennedy proposed another variant of PSO called Fully Informed Particle Swarm (FIPS). In [114], Ratnaweera and Halgamuge proposed another variant of the PSO algorithm called PSO time varying

acceleration coefficient (PSO-TVAC) based on the PSO-TVIW variant. In [174], Higashi et al. introduced the "mutation" concept to the PSO algorithm and proposed a variant called mutation PSO (MPSO). In [114], the authors introduced a variant called mutation PSO with a time varying acceleration coefficient (MPSO-TVAC). In [114], Ratnaweera and Halgamuge introduced self-organizing hierarchical PSO (HPSO). The CSO is proposed to tackle large-scale problems. In [9], although CSO is built on the PSO idea, the neither the local best position nor the global best position is involved in updating the particles' positions. In addition, CSO updates only half of the population in each iteration. The CSO is introduced to tackle large-scale optimization problems to solve problems of high dimension up to 5000. In [115], a modified CSO (MCSO) is proposed as a varaint of CSO where two-thirds of of the population are updated by a tri-competitive criterion. The results show the superiority of MCSO over the basic CSO version. In [116], a variant of CSO is introduced called Inherited Competitive Swarm Optimizer (ICSO). The variant is built on both the human learning principles and the CSO, and the results show better performance than the basic CSO over CEC2008 benchmark problems.

4.2.2 Bacterial Foraging Algorithm

E. coli have a control system that gives them the ability to search for food and avoid noxious regions [7]. In [7], Passino introduced a SI algorithm called bacterial foraging. The algorithm is built on the foraging behavior of *E.coli*. The BF algorithm consists of four phases: *swim* or *tumble*, *chemotactic*, *reproduction*, and *elimination and dispersal*. The **swimming** phase is represented by one or more steps in the same direction as its previous step. The swimming decision is made when a bacterium achieves better fitness value, while the **tumble** (change direction) decision is made when the bacterium receives a worse fitness value. Thus, a bacterium keeps swimming until it reaches the maximum number of swimming steps N_s or it reaches a anxious region, and it is called the **chemotactic** phase. After completing chemotactic steps N_c steps, the fitness value of each bacterium during its lifetime is accumulated. Then, the bacteria are sorted in an ascending order based on the accumulated fitness value. The new population is divided into two equal parts, least healthy bacteria and most healthy bacteria. The least healthy bacteria die, removed from the search space, and most healthy bacteria is split into two bacteria that are placed at the same location, which is called **reproduction** phase. Thus, the number of individual in the population after each reproduction phase is constant. After completing all reproduction N_{re} steps, the **elimination and dispersal** event occurs. In the elimination and dispersal event, each bacterium in the new population is subjected to be eliminated and dispersed with probability P_{ed}.

4.2.3 Imperialist Competitive Algorithm

The ICA is built on the socio-politically imperialism concept where an agent in the population is represented by a country. The agent in the population could be colonies or imperialists where the stronger empires try to colonize the weakest colonies from weaker empires and make them part of their colonies [8]. The ICA was introduced

and evaluated on different benchmark functions, and the results show its ability to tackle different optimization problems. The basic version of ICA was proposed to solve continuous optimization problems such as tuning neural network weights for UCAV global path planning [117]. Later version of ICA was implemented to tackle discrete optimization problem such as Traveling salesman problem (TSP), flowlines scheduling problems (FSP), facility line design problem FLP.

4.2.4 Advantages, Limitations, and Applications of SI Algorithms

SI have many *advantages* over traditional optimization techniques and the main ones are: scalability, adaptability, collective robustness, and individual simplicity [118, 119]. SI algorithms are highly *scalable*, and their control mechanism does not depend on the population size. They are self organized techniques whose individuals interact in direct or indirect ways through the search space. Thus, they have the ability to *adapt* the behavior of individuals in the population to any dynamic changes on the run time. The performance of SI algorithms in tackling different optimization problems proves their *robustness* where there is no single point of failure, but their individuals cooperate and repeat the same behavior. In [120], Elbeltagi et al. compared five algorithms: genetic algorithms, memetic algorithms, particle swarm, ant colony systems, and shuffled frog leaping. The results showed that PSO is the best among all algorithms in terms of computational efficiency. The main advantage of PSO over GA is that it is more simpler, robust, and faster in convergence. In [121], Hassan et al. conducted a number of experiments and concluded that PSO is more computationally efficient than GA. PSO has the ability to control its convergence using its inertia weight better than GA using the rates of crossover and mutation [122]. PSO using small population size performs better than GA using large population size. In [169], Veeramachaneni et al. concluded that PSO is better than GA in solving continuous optimization problem. In [123], the results showed that PSO is better than GA in solving profiled corrugated horn antenna design. In [124], Afandie et al. conducted an experiment to compare between BF and EP in solving optimal load shedding. The results showed that BF outperforms EP in terms of quality of solutions and speed. In [125], Alsariera et al. implemented BF and the bat algorithm on several continuous benchmark functions. The results showed that BF provides more accurate solutions compared to the bat algorithm (BA), but BA exhibits faster convergence rate. In [126], Kamalanand et al. summarized that BF achieved higher efficiency than PSO in computing the optimal dosage of antiretroviral drugs for therapy planning in HIV/AIDS patients. The ICA has a key feature, which is its fast convergence. It has been implemented to tackle different optimization problems [127].

SI algorithms are growing fast, and they offer an alternative way for tackling complex problems, but they have some *limitations and challenges*: parameter tuning, stagnation, and time critical applications. Unlike deterministic methods, stochastic algorithms, including nature-inspired techniques, embed inherent randomness that makes them sensitive to parameters tuning. Parameter tuning for SI and other randomized algorithms is one of the most important but difficult tasks. Regardless of how sensitive a given randomized algorithm is to its random parameters, tuning should

be conducted following an appropriate methodology in order to have a fair comparative assessment of performance. Actually, parameter tuning can be seen as an optimization problem requiring an adequate solving method. There are two main categories of parameters' tuning: off-line and online. Off-line parameter tuning can be done manually following a trial-and-error manner, or automatically [128]. A group of researchers suggested that SI algorithm parameters are predefined in a trial and error manner based on the problem characteristics, which is considered as an old fashion. Automatic parameter tuning can be used to enhance the performance of an algorithm without requiring a prior knowledge. In addition, it prevents us from missing relevant parameters values, which can be the case if we follow a trial-and-error manner. In this regard, in [129, 130], racing techniques use statistical test to exclude parameters' values that achieve lower performance.

Online tuning takes place during the run time period, and can be adaptive/self-adaptive or deterministic [131–134]. This means that a trial start of parameters' values is initiated, and these latter are improved during the run-time of the algorithm. In the online-deterministic technique, the parameters' values are updated based on deterministic rules (such as increasing or decreasing weights in a PSO technique) that are updated every set of iterations. The online-deterministic method is difficult as determining the number of iterations at which the parameters will update is not clear. The self-adaptive method is a process where the parameters' values are updated based on the fitness of solution. A good survey paper on self-adaptive methods can be found in [135].

SI algorithms may suffer from stagnation and convergence to local minima, because they do not have central coordination. In [136], Clerc introduced stagnation analysis for different PSO variants. For instances, in [137], the author concluded that a hybrid algorithm that combined Hopfield neural network and MNC local search outperforms PSO in tackling CSP problem. The ICA also suffers from stagnation problem when it is implemented for high dimension and complex multimodal functions. In [138], Abdi et al. introduced a variant of ICA called GICA to overcome the stagnation problem and the results show a a significant improvement over the basic ICA. In addition, the ICA effectiveness, limitations, and applicability in many domains are investigated [139].

In BF, elimination and dispersal helps in reducing the stagnation behavior [140]. In [141], stagnation occurs in ACO when all ants to follow the same path to reach destination. Sharvani discussed a few way to alleviate stagnation in ACO. In [142], Soundappan introduced a way to avoid stagnation in ABC. In addition, time critical applications require critical decision, control of the system, and acceptable solutions within restricted time frames; such as in underwater sensor networks [143]. SI algorithms are not applicable for time-critical applications, because the solutions of SI algorithms are not predefined. However, they are suitable for non-time critical applications. In [144], Pal et al. compared four different SI algorithms: PSO, ACO, ABC, and FA. The author summarized the advantages and disadvantages of each SI algorithm, and summarized in general the advantages and limitations of SI algorithms.

SI algorithms have been successfully implemented in many applications. In [145], Fornarelli et al. introduced a comprehensive reference of using SI algorithms in the field of electrical and electronic engineering. In addition, Bai et al. introduced a survey

of implementing the SI algorithms in the electric power system. In [18], Karaboga et al. introduced a comprehensive survey about ABC and its applications in electrical, electronic, and control engineering. SI algorithms have been implemented in different applications in mechanical engineering, such as modeling of mechanical properties of as-cast Mg-Li-Al alloys [146], structural damage assessment using FRF [147]. In [18, 19], the reader can find more applications in mechanical engineering. In addition, SI algorithms have been widely used to solve different optimization problems in the civil engineering [19, 148]. Furthermore, they have been used in medical engineering, such as diagnosing the medical diseases, classification of magnetic resonance, parameters adjustment of medical microdevices [19]. In [149], Omran completed his PhD in image processing using PSO. Furthermore, they have been used extensively in biomedical research. The key challenge in biomedical problems is located in the huge amount of their data; therefore, problems require approximate algorithms rather than exact algorithms. In [150], Poli et al surveyed more than 25 different biomedical problems that have been solved using PSO, such as gene selection and cancer classification [151], cancer survival prediction [152], protein structure prediction in the 3D HP model [153], identify transcription factor binding sites [154], drug design [155]. In addition, they have been implemented in communication theory, such as antenna selection in multiple-input-multiple-output (MIMO) system [156], optimizing coverage in indoor Ultra-wideband (UWB) communication system [157], scheduling multichannel and multi-timeslot in time constrained wireless sensor networks [158], and non-linear channel equalization [159]. In [160], a hybrid ICA and GA is implemented to the multi-processor open shop scheduling problem. In [161], the CIA was implemented to design a linear induction motor. In [162], ICA was utilized to optimize the skeletal structures. In [163], the authors proposed a variant of ICA called chaotic ICA, and it was used as image matching approach. In [164], ICA is implemented in solving integrated product mix-outsourcing optimization problem. In [165], the authors implemented ICA for materials property evaluation from indentation test curve. In [166], ICA was implemnted to tune the IPD controller. In [167], ICA was implemented to tackle the scheduling of receiving and shipping trucks in cross-docking systems. In [139], ICA is implemented to tackle the assembly sequence planning problem.

5 Performance Analysis of NI Algorithms

The performance analysis of NI algorithms is a significant task that should be done fairly. In this section, we discuss several guidelines that should be taken into consideration when evaluating a NI algorithm and/or comparing NI algorithms rigorously. Three phases should be implemented when conducting a performance analysis of a NI algorithm rigorously: experimental design, measurement, and reporting [181, 182]. During the experimental design phase, the goals of the experiments are set up, and the input instances are defined. During the measurement phase, all measures to be computed are selected. The results that are obtained by applying different statistical analyses should be reproducible. The reporting phase is the final stage in which the results are presented in a comprehensive way.

5.1 Experimental Design

In the experimental design phase, all goals should be clearly defined as a first step, such as search time quality of solution. The second step is the appropriate selection of the input instances. There are two main types of input instances: real-life instances and constructed instances. Real-life instances are considered the most appropriate benchmarks for a performance evaluation of NI algorithms; however, it is not easy to obtain. On the other hand, constructed instances, also known as standard instances, are available to the public on the internet, and they include well-known instances for continuous optimization or discrete optimization. The main disadvantage of the standard instances is that they do not reflect the level of difficulty of the real-world problems. In continuous optimization, there are several well-known benchmarks, such as Schaffer, Ackley, Griewank, Rastrigin, Rosenbrock,...etc. This group of standard instances has different properties, such as uncorrelated, nonseparable, nonlinear, and nonsymmetric. These properties help to mimic the real-world problems. In addition, NI algorithms have parameters that should be tuned, because those parameters' values have significant influence on the robustness of the algorithms and the obtained quality of solutions. In the performance evaluation process, input instances should be divided into two parts: parameters calibration and performance evaluation. The obtained values of those parameters should be the same for all instances during the experiment.

5.2 Measurements

In the measurement phase, the performance indicators are selected for evaluation, and statistical analysis is applied to obtain the desired results. Exact algorithms guarantee the global optimal solution. Search time is considered the main indicator to evaluate the efficiency of an exact algorithm; however, other indicators beside the search time should be taken into consideration when evaluating NI algorithms, such as quality of solutions, computational effort, and robustness [181]. The quality of solutions is evaluated in terms of precision. Computational effort represents the computation time; it is defined as CPU time including preprocessing and postprocessing time [182]. The number of function evaluations is used often as an indicator for computational effort. Robustness is the insensitivity against small changes in the input instances or NI parameters. It is one of the indicators to measure the performance according to input instances that have different properties.

Given the randomness of the NI techniques and also the problem instances (when randomly generated instances are used). Statistical analysis methods are applied after the results are obtained for different measures. Those methods are used to conduct an assessment of the evaluated NI algorithm. Average and standard deviations are aggregate numbers that should be taken into consideration when using any performance indicator, such as quality of solutions and its associated computational effort. Then, statistical tests are applied to analyze and compare NI algorithms. Statistical tests are used to estimate the confidence of results being scientifically valid, such as *t-tests* and *ANOVA* tests. The *t-test* is applied under normal conditions, and *ANOVA* test is used to compare more than two algorithms.

5.3 Reporting Experimental Evaluation Results

Presenting the results is an important factor that helps researchers gain insight of the experiments. Using tables to present large amounts of data is not sufficient; therefore, visualizing the results (through charts) is considered as a complementary step in understanding the results [183]. The relationship between performance indicators, such as quality of solutions, search time, robustness, and size of instances can be represented graphically. Graphical representation has different forms, such as deviation bars, scatter plots, and interaction plots. The compromise between different performance measures can also be represented using scatter plots, such as the relationship between quality of solutions versus time, or robustness versus time or quality.

6 Conclusion and Future Directions

This work reports on a survey on nature-inspired methods. These techniques are good alternatives, when exact methods fail to solve a given combinatorial problem in a reasonable amount of time. The survey includes some of the most popular NI methods and their significant variants as described in the literature. The advantages and limitations of NI algorithms have been discussed for each category. We support this work with additional valuable references that help the reader to get a better understanding of NI algorithms for real-world applications. The significant number and variety of the applications that have been successfully solved using NI tech demonstrate their efficiency. A well-designed experiment to evaluate a NI algorithm is explained at the end of the survey [184].

Each NI algorithm depends on tunable parameters, and the tuning process is an open area of research. Those parameters influence the complexity of the algorithm and make the analysis process more complex. There are several studies that have been conducted to help resolve this issue [131, 134, 185]. For instance, the parameters may be adaptively tuned during run time. Recent research has concluded that the tuning problem of NI algorithms is somewhat similar to the tuning problem faced in machine learning [186].

One of the main challenges is that the performance of most NI algorithms deteriorates when the dimension and size of the problem increases. Recent research trends involve the development of powerful new NI algorithms to tackle large-scale optimization problem. Thus, scalability for high-dimensional problem becomes a significant factor when proposing new NI algorithms. For instance, cooperative coevolving PSO (CCPSO) is a variant of PSO that was proposed to address the issue of scaling up when solving large-scale optimization problems (up to 2000 real-valued variables). In addition, there is another research trend to enhance the performance of NI algorithms by developing hybrid optimization methods. Many research works have been conducted in the last decade regarding the hybridization of NI algorithms with other algorithms: exact or approximate. In [187], Christian introduced a survey about hybridization of NI algorithms with other algorithms.

In addition, we listed guidelines to be followed by anyone planning to propose a new nature-inspired technique (in addition to sharing the related code, as stated earlier). These guidelines are meant to address the address the "metaphor-based nethodologies" that some researchers are proposing and claiming to be new techniques, while they are actually embedding old ideas taken for known metaheuristics.

In recent years, we are experiencing more and more proposed nature-inspired methods. Unfortunately, some of these new techniques are more of "metaphor-based nethodologies" disguising and embedding old ideas taken for known metaheuristics [188]. In order to address this issue, several guidelines have been discussed in order to refrain from proposing similar methods [182, 188, 189]. We can summarize these guidelines as follows. Any new proposed nature-inspired method should have innovative basic ideas [188]. The components of a proposed algorithm should be well described and evaluated. The relations between these components and those in existing techniques such as GAs and PSOs should be well defined. The proposed method should demonstrate an actual contribution to the field. A rigorous methodology, including experiments demonstrating the merits of the new method, through fair comparative performance, should be conducted. The results, validated through statistical models, should clearly show the superiority of the new technique for some relevant instances.

Scientists concluded that we are in need of a unique publicly available software framework for NI algorithms to reduce the development effort and help compare NI methods [10]. In [189], Burke et al proposed an object oriented framework that is used in evaluating approximate search algorithms. We suggest a framework/platform that includes the code and problem instances for all published algorithms. This platform can be overseen by a dedicated NI committee who will be responsible for collecting and reviewing code and problem instances. We also encourage any researcher who plans to publish a new algorithm or a variant, to submit the related code and instances to the NI committee.

Finally, proposing efficient topologies is an open area of research. Topologies describe how agents in the same population communicate with one another. Many theoretical studies on such topologies have been conducted to improve the performance of different algorithms. Understanding topologies is an important step toward understanding the behavior of different search components in NI algorithms.

Declarations

Conflicts of Interest On behalf of all authors, the corresponding author states that there is no conflict of interestOn behalf of all authors, the corresponding author states that there is no conflict of interest.

References

1. Rao SS (2009) Engineering optimization: theory and practice. John Wiley & Sons
2. Boussaïd I, Lepagnot J, Siarry P (2013) A survey on optimization metaheuristics. Inf Sci 237:82–117

3. Fogel LJ, Owens AJ, Walsh MJ (1966) Artificial intelligence through simulated evolution. Wiley, Chichester, WS, UK
4. Holland JH (1975) Adaptation in natural and artificial systems: an introductory analysis with applications to biology, control, and artificial intelligence
5. Kirkpatrick S, Gelatt CD, Vecchi MP (1983) Optimization by simulated annealing. Science 220(4598):671–680
6. Kennedy J, Eberhart R (1995) Particle swarm optimization. In Proc IEEE Intl Con on Neural Networks (Perth, Australia). pp. 1942–1948
7. Passino KM (2002) Biomimicry of bacterial foraging for distributed optimization and control. IEEE Control Syst 22(3):52–67
8. Atashpaz-Gargari E, Lucas C (2007) Imperialist competitive algorithm: an algorithm for optimization inspired by imperialistic competition. In 2007 IEEE congress on evolutionary computation. pp. 4661–4667
9. Cheng R, Jin Y (2014) A competitive swarm optimizer for large scale optimization. IEEE Transactions on Cybernetics 45(2):191–204
10. Talbi EG (2009) Metaheuristics: from design to implementation, volume 74. John Wiley & Sons
11. Archetti F, Schoen F (1984) A survey on the global optimization problem: general theory and computational approaches. Ann Oper Res 1(2):87–110
12. Fister Jr I, Yang XS, Fister I, Brest J, Fister D (2013) A brief review of nature-inspired algorithms for optimization. arXiv preprint arXiv:1307.4186
13. Talbi EG (2020) Machine learning into metaheuristics: a survey and taxonomy of data-driven metaheuristics
14. Dechter R. (2003) Constraint processing. Morgan Kaufmann
15. Fomin FV, Kratsch D (2010) Exact exponential algorithms. Springer Science & Business Media
16. Applegate D, Bixby R, Cook W, Chvátal V (1998) On the solution of traveling salesman problems. CRPC-TR98744
17. Cheeseman PC, Kanefsky B, Taylor WM (1991) Where the really hard problems are. In IJCAI (91)331–337
18. Karaboga D, Gorkemli B, Ozturk C, Karaboga N (2014) A comprehensive survey: artificial bee colony (abc) algorithm and applications. Artif Intell Rev 42(1):21–57
19. Zhang Y, Wang S, Ji G (2015) A comprehensive survey on particle swarm optimization algorithm and its applications. Math Probl Eng
20. Yang XS (2010) Engineering optimization: an introduction with metaheuristic applications. John Wiley & Sons
21. Xu L, Hutter F, Hoos H, Leyton-Brown K (2012) Evaluating component solver contributions to portfolio-based algorithm selectors. In International Conference on Theory and Applications of Satisfiability Testing. Springer, pp. 228–241
22. Jin Y (2005) A comprehensive survey of fitness approximation in evolutionary computation. Soft Comput 9(1):3–12
23. Zanakis SH, Evans JR (1981) Heuristic "optimization": Why, when, and how to use it. Interfaces 11(5):84–91
24. Crainic TG, Toulouse M (2003) Parallel strategies for meta-heuristics. In Handbook of metaheuristics. Springer, pp. 475–513
25. Szu HH, Hartley RL (1987) Nonconvex optimization by fast simulated annealing. Proceedings of the IEEE 75(11):1538–1540
26. Tsallis C, Stariolo DA (1996) Generalized simulated annealing. Physica A 233(1-2):395–406
27. Creutz M (1983) Microcanonical monte carlo simulation. Phys Rev Lett 50:1411–1414
28. Dueck G, Scheuer T (1990) Threshold accepting: a general purpose optimization algorithm appearing superior to simulated annealing. J Comput Phys 90(1):161–175
29. El Yafrani M, Ahiod B (2016) Population-based vs. single-solution heuristics for the travelling thief problem. In Proceedings of the Genetic and Evolutionary Computation Conference 2016. ACM, pp. 317–324
30. Van Laarhoven PJ, Aarts EH, Lenstra JK (1992) Job shop scheduling by simulated annealing. Oper Res 40(1):113–125
31. Delahaye D, Chaimatanan S, Mongeau M (2019) Simulated annealing: From basics to applications. In Handbook of Metaheuristics. Springer, pp. 1–35
32. Beheshti Z, Shamsuddin SM (2013) A review of population-based meta-heuristic algorithms. Int J Adv Soft Comput Appl 5(1):1–35

33. Gogna A, Tayal A (2013) Metaheuristics: review and application. J Exp Theor Artif Intell 25(4):503–526
34. Stuckman B, Evans G, Mollaghasemi M (1991) Comparison of global search methods for design optimization using simulation. In 1991 Winter Simulation Conference Proceedings. IEEE, pp. 937–944
35. Atkinson AC (1992) A segmented algorithm for simulated annealing. Stat Comput 2(4):221–230
36. Stokes Z, Mandal A, Wong WK (2020) Using differential evolution to design optimal experiments. Chemom Intell Lab Syst 199:103955
37. García-Ródenas R, García-García JC, López-Fidalgo J, Martín-Baos JA, Wong WK (2020) A comparison of general-purpose optimization algorithms for finding optimal approximate experimental designs. Comput Stat Data Anal 144:106844
38. Shi Y, Zhang Z, Wong WK (2019) Particle swarm based algorithms for finding locally and bayesian d-optimal designs. Journal of Statistical Distributions and Applications 6(1):3
39. Mahmudy WF (2016) Improved simulated annealing for optimization of vehicle routing problem with time windows (vrptw). Kursor 7(3)
40. Kose A, Sonmez BA, Balaban M (2017) Simulated annealing algorithm for graph coloring. arXiv preprint arXiv:1712.00709
41. Emden-Weinert T, Proksch M (1999) Best practice simulated annealing for the airline crew scheduling problem. J Heuristics 5(4):419–436
42. Hanafi R, Kozan E (2014) A hybrid constructive heuristic and simulated annealing for railway crew scheduling. Comput Ind Eng 70:11–19
43. Bayram H, Şahin R (2013) A new simulated annealing approach for travelling salesman problem. Mathematical and Computational Applications 18(3):313–322
44. Siarry P, Berthiau G, Durdin F, Haussy J (1997) Enhanced simulated annealing for globally minimizing functions of many-continuous variables. ACM Trans Math Softw (TOMS) 23(2):209–228
45. Connolly DT (1990) An improved annealing scheme for the qap. Eur J Oper Res 46(1):93–100
46. Misevičius A (2003) A modified simulated annealing algorithm for the quadratic assignment problem. Informatica 14(4):497–514
47. Freitas AA (2003) A survey of evolutionary algorithms for data mining and knowledge discovery. In Advances in Evolutionary Computing. Springer, pp 819–845
48. Deb K (1999) An introduction to genetic algorithms. Sadhana 24(4–5):293–315
49. Goldberg DE (1989) Genetic algorithms in search, optimization, and machine learning, addisson-wesley. Reading, MA
50. Coello CC, Pulido GT (2005) Multiobjective structural optimization using a microgenetic algorithm. Struct Multidiscip Optim 30(5):388–403
51. Krishnakumar K (1990) Micro-genetic algorithms for stationary and non-stationary function optimization. In Intelligent Control and Adaptive Systems. International Society for Optics and Photonics 1196:289–297
52. Syswerda G (1989) Uniform crossover in genetic algorithms. In Proceedings of the third international conference on Genetic algorithms. Morgan Kaufmann Publishers, pp 2–9
53. Ono I, Kita H, Kobayashi S (2003) A real-coded genetic algorithm using the unimodal normal distribution crossover. In Advances in Evolutionary Computing. Springer, pp 213–237
54. Ono I, Kita H, Kobayashi S (1999) A robust real-coded genetic algorithm using unimodal normal distribution crossover augmented by uniform crossover: effects of self-adaptation of crossover probabilities. In Proceedings of the 1st Annual Conference on Genetic and Evolutionary Computation-Volume 1. Morgan Kaufmann Publishers Inc., pp 496–503
55. Deb K, Agrawal RB (1995) Simulated binary crossover for continuous search space. Complex Systems 9(2):115–148
56. Sánchez AM, Lozano M, Villar P, Herrera F (2009) Hybrid crossover operators with multiple descendents for real-coded genetic algorithms: Combining neighborhood-based crossover operators. Int J Intell Syst 24(5):540–567
57. Eshelman LJ, Schaffer JD (1993) Real-coded genetic algorithms and interval-schemata. In Foundations of genetic algorithms. Elsevier, vol 2, pp 187–202
58. Takahashi M, Kita H (2001) A crossover operator using independent component analysis for real-coded genetic algorithms. In Proceedings of the 2001 Congress on Evolutionary Computation (IEEE Cat. No. 01TH8546), volume 1, pages 643–649
59. Munteanu C, Lazarescu V (1999) Improving mutation capabilities in a real-coded genetic algorithm. In Workshops on Applications of Evolutionary Computation. Springer, pp 138–149

60. Korejo I, Yang S, Li C (2010) A directed mutation operator for real coded genetic algorithms. In European Conference on the Applications of Evolutionary Computation. Springer, pp 491–500
61. Srinivas N, Deb K (1994) Muiltiobjective optimization using nondominated sorting in genetic algorithms. Evol Comput 2(3):221–248
62. Rechenberg I (1965) Cybernetic solution path of an experimental problem. In Royal Aircraft Establishment Library Translation
63. Schwefel HP (1981) Numerical Optimization of Computer Models. John Wiley & Sons Inc, New York, NY, USA
64. Fogel DB (1991) System Identification Through Simulated Evolution: A Machine Learning Approach to Modeling. Ginn Press
65. Fogel DB (1992) Evolving artificial intelligence. Doctoral Dissertation
66. Fogel DB (1993) Applying evolutionary programming to selected traveling salesman problems. Cybern Syst 24(1):27–36
67. Fogel DB (2006) Evolutionary computation: toward a new philosophy of machine intelligence. John Wiley & Sons, vol 1
68. Yao X, Liu Y (1996) Fast evolutionary programming. Evolutionary Programming 3:451–460
69. Yao X, Liu Y, Lin G (1999) Evolutionary programming made faster. IEEE Trans Evol Comput 3(2):82–102
70. Lee CY, Yao X (2004) Evolutionary programming using mutations based on the lévy probability distribution. IEEE Trans Evol Comput 8(1):1–13
71. Fogel DB (1997) The advantages of evolutionary computation. In BCEC p 1–11
72. Wieland AP (1991) Evolving controls for unstable systems. In Connectionist Models. Elsevier, pp 91–102
73. Schwefel HP (2000) Advantages (and disadvantages) of evolutionary computation over other approaches. Evol Comput 1:20–22
74. Dimopoulos C, Zalzala AMS (2000) Recent developments in evolutionary computation for manufacturing optimization: problems, solutions, and comparisons. IEEE Trans Evol Comput 4(2):93–113
75. Simpson AR, Dandy GC, Murphy LJ (1994) Genetic algorithms compared to other techniques for pipe optimization. J Water Resour Plan Manag 120(4):423–443
76. Chu PC, Beasley JE (1997) A genetic algorithm for the generalised assignment problem. Comput Oper Res 24(1):17–23
77. Alba E, Troya JM et al (1999) A survey of parallel distributed genetic algorithms. Complexity 4(4):31–52
78. Reddy GT, Reddy MP, Lakshmanna K, Rajput DS, Kaluri R, Srivastava G (2020) Hybrid genetic algorithm and a fuzzy logic classifier for heart disease diagnosis. Evol Intell 13(2):185–196
79. Zhou Y, Wang Y, Wang K, Kang L, Peng F, Wang L, Pang J (2020) Hybrid genetic algorithm method for efficient and robust evaluation of remaining useful life of supercapacitors. Appl Energy 260:114169
80. Back T (1996) Evolutionary algorithms in theory and practice: evolution strategies, evolutionary programming, genetic algorithms. Oxford University Press
81. Moslemipour G, Lee TS, Rilling D (2012) A review of intelligent approaches for designing dynamic and robust layouts in flexible manufacturing systems. Int J Adv Manuf Technol 60(1-4):11–27
82. Leung Y, Gao Y, Zong-Ben X (1997) Degree of population diversity-a perspective on premature convergence in genetic algorithms and its markov chain analysis. IEEE Trans Neural Netw 8(5):1165–1176
83. Hrstka O, Kučerová A (2004) Improvements of real coded genetic algorithms based on differential operators preventing premature convergence. Adv Eng Softw 35(3–4):237–246
84. Fogel DB (1995) A comparison of evolutionary programming and genetic algorithms on selected constrained optimization problems. Simulation 64(6):397–404
85. Swayamsiddha S, Thethi HP. Nonlinear system identification using evolutionary computing based training schemes. Int J Comput Appl 975:8887
86. Storn R, Price K (1997) Differential evolution-a simple and efficient heuristic for global optimization over continuous spaces. J Glob Optim 11(4):341–359
87. Li J, Aickelin U (2003) A bayesian optimization algorithm for the nurse scheduling problem. In The 2003 Congress on Evolutionary Computation, 2003. CEC'03., vol 3, pp 2149–2156

88. Larranaga P, Kuijpers CMH, Murga RH, Inza I, Dizdarevic S (1999) Genetic algorithms for the travelling salesman problem: A review of representations and operators. Artif Intell Rev 13(2):129–170
89. Hussain A, Muhammad YS, Sajid MN, Hussain I, Shoukry AM, Gani S (2017) Genetic algorithm for traveling salesman problem with modified cycle crossover operator. Comput Intell Neurosci
90. Davis L (1985) Job shop scheduling with genetic algorithms. In Proceedings of an international conference on genetic algorithms and their applications, vol 140
91. Chan H, Mazumder P, Shahookar K (1991) Macro-cell and module placement by genetic adaptive search with bitmap-represented chromosome. VLSI, 12(1)
92. Boudjelaba K, Ros F, Chikouche D (2014) An efficient hybrid genetic algorithm to design finite impulse response filters. Expert Systems with Applications 41(13):5917–5937
93. Karaboga N, Cetinkaya B (2006) Design of digital fir filters using differential evolution algorithm. Circuits Systems Signal Process 25(5):649–660
94. Karaboga N (2005) Digital iir filter design using differential evolution algorithm. EURASIP Journal on Applied Signal Processing 1269–1276:2005
95. Storn S (1996) Differential evolution design of an iir-filter. In Proceedings of IEEE international conference on evolutionary computation. IEEE, pp 268–273
96. Dasgupta D, Michalewicz Z (2013) Evolutionary algorithms in engineering applications. Springer Science & Business Media
97. Man KF, Tang KS, Kwong Sam (1996) Genetic algorithms: concepts and applications [in engineering design]. IEEE Trans Ind Electron 43(5):519–534
98. Bhoskar MT, Kulkarni OK, Kulkarni NK, Patekar SL, Kakandikar GM, Nandedkar VM (2015) Genetic algorithm and its applications to mechanical engineering: A review. Materials Today: Proceedings 2(4-5):2624–2630
99. Hatanaka T, Uosaki K, Yamada Y (1997) Evolutionary approach to system identification. IFAC Proceedings Volumes 30(11):1327–1332
100. Fahmi M, Samad A (2014) Evolutionary computation in system identification: Review and recommendations. Int J Autom Control pp 208–216
101. Lewin DR (2005) Evolutionary algorithms in control system engineering. IFAC Proceedings Volumes 38(1):45–50
102. Fleming PJ, Purshouse RC (2002) Evolutionary algorithms in control systems engineering: a survey. Control Eng Pract 10(11):1223–1241
103. Alcalá-Fdez J, Sanchez L, Garcia S, del Jesus MJ, Ventura S, Garrell JM, Otero J, Romero C, Bacardit J, Rivas VM et al (2009) Keel: a software tool to assess evolutionary algorithms for data mining problems. Soft Comput 13(3):307–318
104. Bounsaythip C, Alander JT (1997) Genetic algorithms in image processing-a review. In Proceedings of the Third Nordic Workshop on Genetic Algorithms and their Applications (3NWGA) pp 173–192
105. Paulinas M, Ušinskas A (2007) A survey of genetic algorithms applications for image enhancement and segmentation. Inf Tech Control 36(3)
106. Omran MG, Engelbrecht AP, Salman A (2005) Differential evolution methods for unsupervised image classification. In 2005 IEEE Congress on Evolutionary Computation. IEEE 2:966–973
107. Bonabeau E, Marco DD, Dorigo M, Theraulaz G et al (1999) Swarm intelligence: from natural to artificial systems. Oxford University Press, vol 1
108. Eberhart R, Kennedy J (1995) A new optimizer using particle swarm theory. In Micro Machine and Human Science, 1995. MHS'95., Proceedings of the Sixth International Symposium on. IEEE pp 39–43
109. Shi Y, Eberhart RC (1999) Empirical study of particle swarm optimization. In Evolutionary computation, 1999. CEC 99. Proceedings of the 1999 congress on. IEEE 3:1945–1950
110. Shi Y, Eberhart R (1998) A modified particle swarm optimizer. In Evolutionary Computation Proceedings, 1998. IEEE World Congress on Computational Intelligence., The 1998 IEEE International Conference on. IEEE pp 69–73
111. Zheng YL, Ma LH, Zhang LY, Qian JX (2003) Empirical study of particle swarm optimizer with an increasing inertia weight. In Evolutionary Computation, 2003. CEC'03. The 2003 Congress on. IEEE 1:221–226
112. Clerc M, Kennedy J (2002) The particle swarm-explosion, stability, and convergence in a multidimensional complex space. IEEE Trans Evol Comput 6(1):58–73

113. Mendes R, Kennedy J, Neves J (2004) The fully informed particle swarm: simpler, maybe better. IEEE Trans Evol Comput 8(3):204–210
114. Ratnaweera A, Halgamuge SK, Watson HC (2004) Self-organizing hierarchical particle swarm optimizer with time-varying acceleration coefficients. IEEE Trans Evol Comput 8(3):240–255
115. Mohapatra P, Das KN, Roy S (2017) A modified competitive swarm optimizer for large scale optimization problems. Appl Soft Comput 59:340–362
116. Mohapatra P, Das KN, Roy S (2019) Inherited competitive swarm optimizer for large-scale optimization problems. In Harmony Search and Nature Inspired Optimization Algorithms. Springer, pp 85–95
117. Duan H, Huang L (2014) Imperialist competitive algorithm optimized artificial neural networks for ucav global path planning. Neurocomputing 125:166–171
118. Ahmed H, Glasgow J (2012) Swarm intelligence: concepts, models and applications. Queens University Technical Report, School Of Computing
119. Olariu S, Zomaya AY (2005) Handbook of bioinspired algorithms and applications. Chapman and Hall/CRC
120. Elbeltagi E, Hegazy T, Grierson D (2005) Comparison among five evolutionary-based optimization algorithms. Adv Eng Inform 19(1):43–53
121. Hassan R, Cohanim B, De Weck O, Venter G (2005) A comparison of particle swarm optimization and the genetic algorithm. In 46th AIAA/ASME/ASCE/AHS/ASC structures, structural dynamics and materials conference p 1897
122. Rahmat-Samii Y (2003) Genetic algorithm (ga) and particle swarm optimization (pso) in engineering electromagnetics. In 17th International Conference on Applied Electromagnetics and Communications. ICECom IEEE, pp 1–5
123. Diaz L, Milligan TA (1996) Antenna Engineering Using Physical Optics: Practical CAD Techniques and Software, 1st edn. Artech House Inc, Norwood, MA, USA
124. Afandie WN, Rahman TK, Zakaria Z (2016) Comparative analysis of bacterial foraging optimization algorithm and evolutionary programming for load shedding in power system. Int J Simul Syst Sci Technol 17(41)
125. Alsariera YA, Alamri HS, Nasser AM, Majid MA, Zamli KZ (2014) Comparative performance analysis of bat algorithm and bacterial foraging optimization algorithm using standard benchmark functions. In 2014 8th. Malaysian Software Engineering Conference (MySEC). IEEE, pp 295–300
126. Kamalanand K, Jawahar PM (2016) Comparison of particle swarm and bacterial foraging optimization algorithms for therapy planning in hiv/aids patients. Int J Biomath 9(02):1650024
127. Ji X, Gao Q, Yin F, Guo H (2016) An efficient imperialist competitive algorithm for solving the qfd decision problem. Math Probl Eng
128. Huang C, Li Y, Yao X (2019) A survey of automatic parameter tuning methods for metaheuristics. IEEE Trans Evol Comput 24(2):201–216
129. Birattari M, Stützle T, Paquete L, Varrentrapp K et al (2002) A racing algorithm for configuring metaheuristics. In Gecco, vol 2
130. Yuan B, Gallagher M (2004) Statistical racing techniques for improved empirical evaluation of evolutionary algorithms. In International Conference on Parallel Problem Solving from Nature. Springer, pp 172–181
131. Chen L, Xu X, Chen YX (2004) An adaptive ant colony clustering algorithm. In Proceedings of 2004 International Conference on Machine Learning and Cybernetics (IEEE Cat. No. 04EX826), vol 3, pp 1387–1392
132. Chen H, Zhu Y, Hu K (2011) Adaptive bacterial foraging optimization. In Abstract and Applied Analysis. Hindawi, vol 2011
133. Dasgupta S, Das S, Abraham A, Biswas A (2009) Adaptive computational chemotaxis in bacterial foraging optimization: an analysis. IEEE Trans Evol Comput 13(4):919–941
134. Shi Y, Eberhart RC (2001) Fuzzy adaptive particle swarm optimization. In Proceedings of the 2001 congress on evolutionary computation (IEEE Cat. No. 01TH8546), vol 1, pp 101–106
135. Bäck T (2001) Introduction to the special issue: Self-adaptation. Evol Comput 9(2):3–4
136. Clerc M (2006) Stagnation analysis in particle swarm optimisation or what happens when nothing happens. Tech Rep
137. Bouhouch A, Loqman C, Bennis H, El Qadi A (2019) A comparative study of chn-mnc, ga and pso for solving constraints satisfaction problems. In Third International Conference on Computing and Wireless Communication Systems, ICCWCS 2019. European Alliance for Innovation (EAI)

138. Abdi Y, Lak M, Seyfari Y (2017) Gica: Imperialist competitive algorithm with globalization mechanism for optimization problems. Turk J Electr Eng Comput Sci 25(1):209–221
139. Zhou W, Yan J, Li Y, Xia C, Zheng J (2013) Imperialist competitive algorithm for assembly sequence planning. Int J Adv Manuf Technol 67(9–12):2207–2216
140. Vijay R (2012) Intelligent bacterial foraging optimization technique to economic load dispatch problem. International Journal of Soft Computing and Engineering (IJSCE) 2(2):2231–2307
141. Sharvani GS, Ananth AG, Rangaswamy TM (2012) Analysis of different pheromone decay techniques for aco based routing in ad hoc wireless networks. Int J Comput Appl 56(2)
142. Jagadeesh S, Sugumar R (2017) A comparative study on artificial bee colony with modified abc algorithm. European Journal of Applied Sciences 9(5):243–248
143. Zhou Z, Peng Z, Cui JH, Shi Z (2010) Efficient multipath communication for time-critical applications in underwater acoustic sensor networks. IEEE/ACM Trans Networking 19(1):28–41
144. Pal NS, Sharma S (2013) Robot path planning using swarm intelligence: A survey. Int J Comput Appl 83(12):5–12
145. Fornarelli G (2012) Swarm intelligence for electric and electronic engineering. IGI Global
146. Ming L, Hai H, Aimin Z, Yingde S, Zhao L, Xingguo Z (2012) Modeling of mechanical properties of as-cast mg-li-al alloys based on pso-bp algorithm. China Foundry 9(2)
147. Mohan SC, Maiti DK, Maity D (2013) Structural damage assessment using frf employing particle swarm optimization. Appl Math Comput 219(20):10387–10400
148. Lu P, Chen S, Zheng Y (2012) Artificial intelligence in civil engineering. Math Probl Eng
149. Omran MGH et al (2004) Particle swarm optimization methods for pattern recognition and image processing. PhD thesis, Citeseer
150. Poli R (2007) An analysis of publications on particle swarm optimization applications. Department of Computer Science, University of Essex, Essex, UK
151. Saraswathi S, Sundaram S, Sundararajan N, Zimmermann M, Nilsen-Hamilton M (2011) Icga-pso-elm approach for accurate multiclass cancer classification resulting in reduced gene sets in which genes encoding secreted proteins are highly represented. IEEE/ACM Transactions on Computational Biology and Bioinformatics (TCBB) 8(2):452–463
152. Xu R, Cai X, Wunsch DC (2006) Gene expression data for dlbcl cancer survival prediction with a combination of machine learning technologies. In 2005 IEEE Engineering in Medicine and Biology 27th Annual Conference pp 894–897
153. Mansour N, Kanj F, Khachfe H (2012) Particle swarm optimization approach for protein structure prediction in the 3d hp model. Interdisciplinary Sciences: Computational Life Sciences 4(3):190–200
154. Karabulut M, Ibrikci T (2012) A bayesian scoring scheme based particle swarm optimization algorithm to identify transcription factor binding sites. Appl Soft Comput 12(9):2846–2855
155. Cedefto W, Agraflotis D (2005) Particle swarms for drug design. In 2005 IEEE Congress on Evolutionary Computation, vol 2, pp 1218–1225
156. Yongqiang H, Wentao L, Xiaohui L (2013) Particle swarm optimization for antenna selection in mimo system. Wirel Pers Commun 68(3):1013–1029
157. Chiu CC, Ho MH, Liao S (2013) Pso and apso for optimizing coverage in indoor uwb communication system. Int J RF Microwave Comput Aided Eng 23(3):300–308
158. Kim YG, Lee MJ (2014) Scheduling multi-channel and multi-timeslot in time constrained wireless sensor networks via simulated annealing and particle swarm optimization. IEEE Commun Mag 52(1):122–129
159. Das G, Pattnaik PK, Padhy SK (2014) Artificial neural network trained by particle swarm optimization for non-linear channel equalization. Expert Systems with Applications 41(7):3491–3496
160. Goldansaz SM, Jolai F, Anaraki AHZ (2013) A hybrid imperialist competitive algorithm for minimizing makespan in a multi-processor open shop. Appl Math Model 37(23):9603–9616
161. Lucas C, Nasiri-Gheidari Z, Tootoonchian F (2010) Application of an imperialist competitive algorithm to the design of a linear induction motor. Energy Convers Manag 51(7):1407–1411
162. Kaveh A, Talatahari S (2010) Optimum design of skeletal structures using imperialist competitive algorithm. Comput Struct 88(21–22):1220–1229
163. Duan H, Chunfang X, Liu S, Shao S (2010) Template matching using chaotic imperialist competitive algorithm. Pattern Recogn Lett 31(13):1868–1875
164. Nazari-Shirkouhi S, Eivazy H, Ghodsi R, Rezaie K, Atashpaz-Gargari E (2010) Solving the integrated product mix-outsourcing problem using the imperialist competitive algorithm. Expert Systems with Applications 37(12):7615–7626

165. Biabangard-Oskouyi A, Atashpaz-Gargari E, Soltani N, Lucas C (2009) Application of imperialist competitive algorithm for materials property characterization from sharp indentation test. International Journal of Engineering Simulation 10(1):11–12
166. Rajabioun R, Hashemzadeh F, Atashpaz-Gargari E, Mesgari B, Rajaei Salmasi F (2008) Identification of a mimo evaporator and its decentralized pid controller tuning using colonial competitive algorithm. In be presented in IFAC World Congress
167. Forouharfard S, Zandieh M (2010) An imperialist competitive algorithm to schedule of receiving and shipping trucks in cross-docking systems. Int J Adv Manuf Technol 51(9-12):1179–1193
168. Alba E, Chicano JF (2006) Evolutionary algorithms in telecommunications. In MELECON 2006-2006 IEEE Mediterranean Electrotechnical Conference, pp 795–798
169. Veeramachaneni K, Peram T, Mohan C, Osadciw LA (2003) Optimization using particle swarms with near neighbor interactions. In Genetic and Evolutionary Computation Conference. Springer, pp 110–121
170. Chaimatanan S, Delahaye D, Mongeau M (2014) A hybrid metaheuristic optimization algorithm for strategic planning of 4d aircraft trajectories at the continental scale. IEEE Comput Intell Mag 9(4):46–61
171. Flores SD, Cegla BB, Cáceres DB (2003) Telecommunication network design with parallel multiobjective evolutionary algorithms. LANC 3:3–5
172. Fogel DB (2000) Evolutionary computation: principles and practice for signal processing. SPIE Press, vol 43
173. Fogel DB, Fogel LJ, Atmar JW (1991) Meta-evolutionary programming. In [1991] Conference Record of the Twenty-Fifth Asilomar Conference on Signals, Systems & Computers. pp 540–545
174. Higashi N, Iba H (2003) Particle swarm optimization with gaussian mutation. In Proceedings of the 2003 IEEE Swarm Intelligence Symposium. SIS'03 (Cat. No. 03EX706). IEEE, pp 72–79
175. Ilonen J, Kamarainen JK, Lampinen J (2003) Differential evolution training algorithm for feedforward neural networks. Neural Process Lett 17(1):93–105
176. Miller JF, Job D, Vassilev VK (2000) Principles in the evolutionary design of digital circuits–part i. Genet Program Evolvable Mach 1(1-2):7–35
177. Wong DF, Leong HW, Liu HW (2012) Simulated annealing for VLSI design. Springer Science & Business Media, vol 42
178. Yao X (1999) Evolving artificial neural networks. Proceedings of the IEEE 87(9):1423–1447
179. Yao X, Liu Y (1997) A new evolutionary system for evolving artificial neural networks. IEEE Trans Neural Netw 8(3):694–713
180. Zebulum RS, Pacheco MA, Be Vellasco MM (2018) Evolutionary electronics: automatic design of electronic circuits and systems by genetic algorithms. CRC Press
181. Barr RS, Golden BL, Kelly JP, Resende MGC, Stewart WR (1995) Designing and reporting on computational experiments with heuristic methods. J Heuristics 1(1):9–32
182. Hooker JN (1995) Testing heuristics: We have it all wrong. J Heuristics 1(1):33–42
183. Tufte ER (2001) The visual display of quantitative information. Graphics press Cheshire, CT, vol 2
184. Chiarandini M, Paquete L, Preuss M, Ridge E (2007) Experiments on metaheuristics: Methodological overview and open issues. Tech Rep DMF-2007-03-003
185. Clerc M (1999) The swarm and the queen: towards a deterministic and adaptive particle swarm optimization. In Proceedings of the 1999 congress on evolutionary computation-CEC99 (Cat. No. 99TH8406). IEEE, vol 3, pp 1951–1957
186. Birattari M, Kacprzyk J (2009) Tuning metaheuristics: a machine learning perspective. Springer, vol 197
187. Blum C, Puchinger J, Raidl GR, Roli A (2011) Hybrid metaheuristics in combinatorial optimization: A survey. Appl Soft Comput 11(6):4135–4151
188. Sörensen K (2015) Metaheuristics–the metaphor exposed. Int Trans Oper Res 22(1), 3–18
189. Burke EK, Curtois T, Kendall G, Hyde M, Ochoa G, Vazquez-Rodriguez JA (2009) Towards the decathlon challenge of search heuristics. In Proceedings of the 11th Annual Conference Companion on Genetic and Evolutionary Computation Conference: Late Breaking Papers. pp 2205–2208