Metaheuristic Techniques

Sunith Bandaru[a], Kalyanmoy Deb[b]

[a]*School of Engineering Science, University of Skövde, Skövde 541 28, Sweden*
[b]*Department of Electrical and Computer Engineering, Michigan State University, East Lansing, 428 S. Shaw Lane, 2120 EB, MI 48824, USA*

COIN Report Number 2016029*

Abstract

Most real-world search and optimization problems involve complexities such as non-convexity, non-linearities, discontinuities, mixed nature of variables, multiple disciplines and large dimensionality, a combination of which renders classical provable algorithms to be either ineffective, impractical or inapplicable. There do not exist any known mathematically motivated algorithms for finding the optimal solution for all such problems in a limited computational time. Thus, in order to solve such problems to practicality, search and optimization algorithms are usually developed using certain *heuristics* that though lacking in strong mathematical foundations, are nevertheless good at reaching an approximate solution in a reasonable amount of time. These so-called metaheuristic methods do not guarantee finding the exact optimal solution, but can lead to a near-optimal solution in a computationally efficient manner. Due to this practical appeal combined with their ease of implementation, metaheuristic methodologies are gaining popularity in several application domains. Most metaheuristic methods are stochastic in nature and mimic a natural, physical or biological principle resembling a search or an optimization process. In this paper, we discuss a number of such methodologies, specifically evolutionary algorithms, such as genetic algorithms and evolution strategy, particle swarm optimization, ant colony optimization, bee colony optimization, simulated annealing and a host of other methods. Many metaheuristic methodologies are being proposed by researchers all over the world on a regular basis. It therefore becomes important to unify them to understand common features of different metaheuristic methods and simultaneously to study fundamental differences between them. Hopefully, such endeavors will eventually allow a user to choose the most appropriate metaheuristic method for the problem at hand.

1. Introduction

The present paper does not claim to be a thorough survey of all existing metaheuristic methods and their related aspects. Instead, the purpose is to describe some of the most popular methods and provide pseudocodes to enable beginners to easily implement them. Therefore, this paper should be seen more as a quick-start guide to popular metaheuristics than as a survey of methods and applications. Readers interested in the history, variants, empirical analysis, specific applications, tuning and parallelization of metaheuristic methods should refer to the following texts:

1. **Books:** [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13]
2. **Surveys:** [14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24]
3. **Combinatorial Problems:** [25, 26, 27, 28, 29, 30, 31, 32, 33, 34]
4. **Analysis:** [35, 36, 37, 38, 39, 40, 41, 42, 43]

*Published in *Decision Sciences: Theory and Practice, Edited by Raghu Nandan Sengupta, Aparna Gupta and Joydeep Dutta, 693–750, CRC Press, Taylor & Francis Group.*

Email addresses: sunith.bandaru@his.se (Sunith Bandaru), kdeb@egr.msu.edu (Kalyanmoy Deb)

5. **Parallelization:** [44, 45, 46, 47, 48, 49, 50, 51, 52].

The development of new metaheuristic methods has picked up pace over the last 20 years. Nowadays, many conferences, workshops, symposiums and journals accept submissions related to this topic. Some of them (in no particular order) are:

Conferences/Symposiums:

1. Genetic and Evolutionary Computation Conference (GECCO)
2. IEEE Congress on Evolutionary Computation (CEC)
3. Evolutionary Multi-criterion Optimization (EMO)
4. Parallel Problem Solving from Nature (PPSN)
5. Foundations of Genetic Algorithms (FOGA)
6. Simulated Evolution And Learning (SEAL)
7. Learning and Intelligent OptimizatioN (LION)
8. International Joint Conference on Computational Intelligence (IJCCI)
9. International Conference on Artificial Intelligence and Soft Computing (ICAISC)
10. IEEE Symposium Series on Computational Intelligence (SSCI)
11. IEEE Swarm Intelligence Symposium (SIS)

Journals:

1. IEEE Transactions on Evolutionary Computation (IEEE Press)
2. Applied Soft Computing (Elsevier)
3. Computers & Operations Research (Elsevier)
4. European Journal of Operational Research (Elsevier)
5. Information Sciences (Elsevier)
6. Evolutionary Computation (MIT Press)
7. Computational Optimization and Applications (Springer)
8. Soft Computing (Springer)
9. Engineering Optimization (Taylor & Francis)
10. IEEE Transactions on Systems, Mans and Cybernetics (IEEE Press)
11. Engineering Applications of Artificial Intelligence (Elsevier)
12. International Transactions in Operational Research (Wiley)
13. Intelligent Automation & Soft Computing (Taylor & Francis)
14. Applied Computational Intelligence and Soft Computing (Hindawi)
15. Journal of Multi-Criteria Decision Analysis (Wiley)
16. Artificial Life (MIT Press)
17. Journal of Mathematical Modelling and Algorithms in Operations Research (Springer, Discontinued)
18. International journal of Artificial Intelligence & Applications (AIRCC)

Some publications that are entirely dedicated to metaheuristics also exist, namely,

1. Journal of Heuristics (Springer)
2. Swarm and Evolutionary Computation (Elsevier)
3. Swarm Intelligence (Springer)
4. Natural Computing (Springer)
5. Genetic Programming and Evolvable Machines (Springer)
6. International Journal of Metaheuristics (Inderscience)
7. International Journal of Bio-Inspired Computation (Inderscience)
8. Memetic Computing (Springer)
9. International Journal of Applied Metaheuristic Computing (IGI Global)
10. Computational Intelligence and Metaheuristic Algorithms with Applications (Hindawi)
11. European Event on Bio-Inspired Computation (EvoStar Conference)

12. International Conference on Adaptive & Natural Computing Algorithms (ICANNGA)
13. Ant Colony Optimization and Swarm Intelligence (ANTS Conference)
14. Swarm, Evolutionary and Memetic Computing Conference (SEMCCO)
15. Bio-Inspired Computing: Theories and Applications (BICTA)
16. Nature and Biologically Inspired Computing (NaBIC)
17. International Conference on Soft Computing for Problem Solving (SocProS)
18. Metaheuristics International Conference (MIC)
19. International Conference on Metaheuristics and Nature Inspired Computing (META Conference)

Implementation of metaheuristic methods, though mostly straightforward, can be a tedious task. Luckily, several software frameworks are freely available on the internet which can be used by beginners to get started with solving their optimization problems. Notable examples are:

1. PISA: A platform and programming language independent interface for search algorithms (`http://www.tik.ee.ethz.ch/pisa/`)
2. ParadisEO: A C++ software framework for metaheuristics (`http://paradiseo.gforge.inria.fr/`)
3. Open BEAGLE: A C++ evolutionary computation framework (`https://code.google.com/p/beagle/`)
4. Evolving Objects: An evolutionary computation framework (`http://eodev.sourceforge.net/`)
5. GAlib: A C++ library of genetic algorithm components (`http://lancet.mit.edu/ga/`)
6. METSlib: A metaheuristic modeling framework and optimization toolkit in C++ (`https://projects.coin-or.org/metslib`)
7. ECF: A C++ evolutionary computation framework (`http://gp.zemris.fer.hr/ecf/`)
8. HeuristicLab: A framework for heuristic and evolutionary algorithms (`http://dev.heuristiclab.com/`)
9. ECJ: A Java-based evolutionary computation research system (`https://cs.gmu.edu/~eclab/projects/ecj/`)
10. jMetal: Metaheuristic algorithms in Java (`http://jmetal.sourceforge.net/`)
11. MOEA Framework: A free and open source Java framework for multiobjective optimization (`http://www.moeaframework.org/`)
12. JAMES: A Java metaheuristics search framework (`http://www.jamesframework.org/`)
13. Watchmaker Framework: An object-oriented framework for evolutionary/genetic algorithms in Java (`http://watchmaker.uncommons.org/`)
14. Jenetics: An evolutionary algorithm library written in Java (`http://jenetics.io/`)
15. Pyevolve: A complete genetic algorithm framework in Python (`http://pyevolve.sourceforge.net/`)
16. DEAP: Distributed evolutionary algorithms in Python (`https://github.com/DEAP/deap`)

These implementations provide the basic framework required to run any of the several available metaheuristics. Software implementations specific to individual methods are also available in plenty. For example, genetic programming (discussed later in Section 10) which requires special solution representation schemes, has several variants in different programming languages, each handling the representation in a unique way.

This paper is arranged as follows. In the next section, we discuss basic concepts and classification of metaheuristics. We also lay down a generic metaheuristic framework which covers most of the available

methods. From Section 3 to Section 14 we cover several popular metaheuristic techniques with complete pseudocodes. We discuss the methods in alphabetical order in order to avoid preconceptions about performance, generality and applicability, thus respecting the No Free Lunch theorem [53] of optimization. In Section 15, we enumerate several less popular methods with a brief description of their origins. These methodologies are ordered by their number of Google Scholar citations. We conclude this paper in Section 16 with a few pointers to future research directions.

2. Concepts of Metaheuristic Techniques

The word 'heuristic' is defined in the context of computing as a method of denoting a rule-of-thumb for solving a problem without the exhaustive application of a procedure. In other words, a heuristic method is one that (i) looks for an approximate solution, (ii) need not particularly have a mathematical convergence proof, and (iii) does not explore each and every possible solution in the search space before arriving at the final solution, hence is computationally efficient.

A metaheuristic method is particularly relevant in the context of solving search and optimization problems. It describes a method that uses one or more heuristics and therefore inherits all the three properties mentioned above. Thus, a metaheuristic method (i) seeks to find a near-optimal solution, instead of specifically trying to find the exact optimal solution, (ii) usually has no rigorous proof of convergence to the optimal solution, and (iii) is usually computationally faster than exhaustive search. These methods are iterative in nature and often use stochastic operations in their search process to modify one or more initial candidate solutions (usually generated by random sampling of the search space). Since many real-world optimization problems are complex due to their inherent practicalities, classical optimization algorithms may not always be applicable and may not fair well in solving such problems in a pragmatic manner. Realizing this fact and without disregarding the importance of classical algorithms in the development of the field of search and optimization, researchers and practitioners sought for metaheuristic methods so that a near-optimal solution can be obtained in a computationally tractable manner, instead of waiting for a provable optimization algorithm to be developed before attempting to solve such problems. The ability of the metaheuristic methods to handle different complexities associated with practical problems and arrive at a reasonably acceptable solution is the main reason for the popularity of metaheuristic methods in the recent past.

Most metaheuristic methods are motivated by natural, physical or biological principles and try to mimic them at a fundamental level through various operators. A common theme seen across all metaheuristics is the balance between *exploration* and *exploitation*. Exploration refers to how well the operators diversify solutions in the search space. This aspect gives the metaheuristic a global search behavior. Exploitation refers to how well the operators are able to utilize the information available from solutions from previous iterations to intensify search. Such intensification gives the metaheuristic a local search characteristic. Some metaheuristics tend to be more explorative than exploitative, while some others do the opposite. For example, the primitive method of randomly picking solutions for a certain number of iterations, represents a completely explorative search. On the other hand, hill climbing where the current solution is incrementally modified until it improves, is an example of completely exploitative search. More commonly, metaheuristics allow this balance between diversification and intensification to be adjusted by the user through operator parameters.

The characteristics described above give metaheuristics certain advantages over the classical optimization methods, namely,

1. Metaheuristics can lead to *good enough* solutions for computationally easy (technically, P class) problems with large input complexity, which can be a hurdle for classical methods.
2. Metaheuristics can lead to *good enough* solutions for the NP-hard problems, i.e. problems for which no known exact algorithm exists that can solve them in a reasonable amount of time.
3. Unlike most classical methods, metaheuristics require no gradient information and therefore can be used with non-analytic, black-box or simulation-based objective functions.

4. Most metaheuristics have the ability to recover from local optima due to inherent stochasticity or deterministic heuristics specifically meant for this purpose.
5. Because of the ability to recover from local optima, metaheuristics can better handle uncertainties in objectives.
6. Most metaheuristics can handle multiple objectives with only a few algorithmic changes.

2.1. Optimization Problems

A non-linear programming (NLP) problem involving n real or discrete (integer, boolean or otherwise) variables or a combination thereof is stated as follows:

$$\begin{array}{rll} \text{Minimize} & f(\mathbf{x}), & \\ \text{subject to} & g_j(\mathbf{x}) \geq 0, & j = 1, 2, \ldots, J, \\ & h_k(\mathbf{x}) = 0, & k = 1, 2, \ldots, K, \\ & x_i^{(L)} \leq x_i \leq x_i^{(U)}, & i = 1, 2, \ldots, n, \end{array} \tag{1}$$

where $f(\mathbf{x})$ is the objective function to be optimized, $g_j(\mathbf{x})$ represent J inequality constraints and $h_k(\mathbf{x})$ represent K equality constraints. Typically, equality constraints are handled by converting them into soft inequality constraints. A survey of constraint handling methods can be found in [54, 55]. A solution $\mathbf{x}$ is feasible if all $J + K$ constraints and variable bounds $[x_i^{(L)}, x_i^{(U)}]$ are satisfied.

The nature of the optimization problem plays an important role in determining the optimization methodology to be used. Classical optimization methods should always be the first choice for solving convex problems. An NLP problem is said to be convex if and only if, (i) f is convex, (ii) all $g_j(\mathbf{x})$ are concave, and (iii) all $h_k(\mathbf{x})$ are linear. Unfortunately, most real-world problems are non-convex and NP-hard, which makes metaheuristic techniques a popular choice.

Metaheuristics are especially popular for solving combinatorial optimization problems because not many classical methods can handle the kind of variables that they involve. A typical combinatorial optimization problem involves an n-dimensional permutation $\mathbf{p}$ in which every entity appears only once:

$$\begin{array}{rll} \text{Minimize} & f(\mathbf{p}), & \\ \text{subject to} & g_j(\mathbf{p}) \geq 0, & j = 1, 2, \ldots, J \\ & h_k(\mathbf{p}) \geq 0, & k = 1, 2, \ldots, K. \end{array} \tag{2}$$

Again, a candidate permutation $\mathbf{p}$ is feasible only when all J constraints are satisfied. Many practical problems involve combinatorial optimization. Examples include knapsack problems, bin-packing, network design, traveling salesman, vehicle routing, facility location and scheduling.

When multiple objectives are involved that conflict with each other, no single solution exists that can simultaneously optimize all the objectives. Rather, a multitude of optimal solutions are possible which provide a trade-off between the objectives. These solutions are known as the Pareto-optimal solutions and they lie on what is known as the Pareto-optimal front. A candidate solution $\mathbf{x_1}$ is said to *dominate* $\mathbf{x_2}$ and denoted as $\mathbf{x_1} \prec \mathbf{x_2}$ if and only if the following conditions are satisfied:

1. $f_i(\mathbf{x_1}) \leq f_i(\mathbf{x_2})\ \forall i \in \{1, 2, \ldots, m\}$,
2. and $\exists j \in \{1, 2, \ldots, m\}$ such that $f_j(\mathbf{x_1}) < f_j(\mathbf{x_2})$.

If only the first of these conditions is satisfied then $\mathbf{x_1}$ is said to *weakly dominate* $\mathbf{x_2}$ and is denoted as $\mathbf{x_1} \preceq \mathbf{x_2}$. If neither $\mathbf{x_1} \preceq \mathbf{x_2}$ nor $\mathbf{x_2} \preceq \mathbf{x_1}$, then $\mathbf{x_1}$ and $\mathbf{x_2}$ are said to be non-dominated with respect to each other and denoted as $\mathbf{x_1}||\mathbf{x_2}$. A feasible solution $\mathbf{x}^*$ is said to be Pareto-optimal if there does not exist any other feasible $\mathbf{x}$ such that $\mathbf{x} \prec \mathbf{x}^*$. The set of all such $\mathbf{x}^*$ (which are non-dominated with respect to each other) is referred to as the Pareto-optimal set. Multi-objective optimization poses additional challenges to metaheuristics due to this concept of non-dominance.

In this paper, we will restrict ourselves to describing metaheuristics in the context of single-objective optimization. Most metaheuristics can be readily used for multi-objective optimization by simply considering Pareto-dominance or other suitable selection mechanisms when comparing different solutions

and by taking extra measures for preserving solution diversity. However, it is important to note that since multi-objective optimization problems with conflicting objectives have multiple optimal solutions, metaheuristics that use a 'population' of solutions are preferred so that the entire Pareto-optimal front can be represented simultaneously.

2.2. Classification of Metaheuristic Techniques

The most common way of classifying metaheuristic techniques is based on the number of initial solutions that are modified in subsequent iterations. Single-solution metaheuristics start with one initial solution which gets modified iteratively. Note that the modification process itself may involve more than one solution, but only a single solution is used in each following iteration. Population-based metaheuristics use more than one initial solution to start optimization. In each iteration, multiple solutions get modified, and some of them make it to the next iteration. Modification of solutions is done through operators that often use special statistical properties of the population. The additional parameter for size of the population is set by the user and usually remains constant across iterations.

Another way of classifying metaheuristics is through the domain that they mimic. Umbrella terms such as bio-inspired and nature-inspired are often used for metaheuristics. However, they can be further sub-categorized as evolutionary algorithms, swarm-intelligence based algorithms and physical phenomenon based algorithms. Evolutionary algorithms (like genetic algorithms, evolution strategies, differential evolution, genetic programming, evolutionary programming, etc.) mimic various aspects of evolution in nature such as survival of the fittest, reproduction and genetic mutation. Swarm-intelligence algorithms mimic the group behavior and/or interactions of living organisms (like ants, bees, birds, fireflies, glow-worms, fishes, white-blood cells, bacteria, etc.) and non-living things (like water drops, river systems, masses under gravity, etc.). The rest of the metaheuristics mimic various physical phenomena like annealing of metals, musical aesthetics (harmony), etc. A fourth sub-category may be used to classify metaheuristics whose source of inspiration is unclear (like tabu search and scatter search) or those that are too few in number to have a category for themselves.

Other popular ways of classifying metaheuristics are [1, 35]:

1. Deterministic versus stochastic methods: Deterministic methods follow a definite trajectory from the random initial solution(s). Therefore, they are sometimes referred to as trajectory methods. Stochastic methods (also discontinuous methods) allow probabilistic *jumps* from the current solution(s) to the next.
2. Greedy versus non-greedy methods: Greedy algorithms usually search in the neighborhood of the current solution and immediately move to a better solution when its found. This behavior often leads to a local optimum. Non-greedy methods either hold out for some iterations before updating the solution(s) or have a mechanism to back track from a local optimum. However, for convex problems, a greedy behavior is the optimum strategy.
3. Memory usage versus memoryless methods: Memory-based methods maintain a record of past solutions and their trajectories and use them to direct search. A popular memory method called tabu search will be discussed in Section 14.
4. One versus various neighborhood methods: Some metaheuristics like simulated annealing and tabu search only allow a limited set of moves from the current solution. But many metaheuristics employ operators and parameters to allow multiple neighborhoods. For example, particle swarm optimization achieves this through various swarm topologies.
5. Dynamic versus static objective function: Metaheuristics that update the objective function depending on the current requirements of search are classified as dynamic. Other metaheuristics simply use their operators to control search.

Most metaheuristics are population-based, stochastic, non-greedy and use a static objective function.

In this paper we avoid making any attempt at classifying individual metaheuristic techniques because of two reasons:

1. Single-solution metaheuristics can often be transformed into population-based ones (and vice versa) either by hybridization with other methods or, sometimes, simply by consideration of multiple initial solutions.
2. The large number of metaheuristics that have been derived off late from existing techniques use various modifications in their operators. Sometimes these variants blur the extent of mimicry so much that referring to them by their domain of inspiration becomes unreasonable.

Therefore, for each metaheuristic, we discuss the most popular and generic variant.

2.3. A Generic Metaheuristic Framework

Most metaheuristics follow a similar sequence of operations and therefore can be defined within a common generic framework as shown in Algorithm 1.

Algorithm 1 A Generic Metaheuristic Framework

Require: SP, GP, RP, UP, TP, N (≥ 1), μ ($\leq N$), λ, ρ ($\leq N$)
1: Set iteration counter $t = 0$
2: Initialize N solutions S_t randomly
3: Evaluate each member of S_t
4: Mark the best solution of S_t as $\mathbf{x}_t^*$
5: **repeat**
6: Choose μ solutions (set P_t) from S_t using a *selection plan (SP)*
7: Generate λ new solutions (set C_t) from P_t using a *generation plan (GP)*
8: Choose ρ solutions (set R_t) from S_t using a *replacement plan (RP)*
9: Update S_t by replacing R_t using ρ solutions from a pool of any combination of at most three sets P_t, C_t, and R_t using an *update plan (UP)*
10: Evaluate each member of S_t
11: Identify the best solution of S_t and update $\mathbf{x}_t^*$
12: $t \leftarrow t + 1$
13: **until** (a *termination plan (TP)* is satisfied)
14: Declare the near-optimal solution as $\mathbf{x}_t^*$

The `REQUIRE` statement indicates that in order to prescribe a metaheuristic algorithm, a minimum of five plans (SP, GP, RP, UP, TP) and four parameters (N, μ, λ, ρ) are needed. The individual plans may involve one or more parameters of their own. Step 1 initializes the iteration counter (t) to zero. It is clear from the repeat-until loop of the algorithm that a metaheuristic algorithm is by nature an iterative procedure.

Step 2 creates an initial set of N solutions in the set S_t. Mostly, the solutions are created at random in between the prescribed lower and upper bounds of variables. For combinatorial optimization problems, random permutations of entities can be created. When problem specific knowledge of good initial solutions is available, they may be included along with a few random solutions to allow exploration.

Step 3 evaluates each of N set members by using the supplied objective and constraint functions. Constraint violation, if any, must be accounted for here to provide an evaluation scheme which will be used in subsequent steps of the algorithm. One way to define constraint violation $CV(\mathbf{x}/\mathbf{p})$ is given by:

$$CV(\mathbf{x}/\mathbf{p}) = \sum_{j=1}^{J} \langle g_j(\mathbf{x}/\mathbf{p}) \rangle, \tag{3}$$

where $\langle \alpha \rangle$ is $|\alpha|$ if $\alpha < 0$, and is zero, otherwise. Before adding the constraint values of different constraints, they need to be normalized [56]. The objective function value $f(\mathbf{x})$ and constraint violation $CV(\mathbf{x}/\mathbf{p})$ value can both be sent to subsequent steps for a true evaluation of the solutions.

Step 4 chooses the best member of the set S_t and saves it as $\mathbf{x}_t^*$. This step requires a pairwise comparison of set members. One way to compare two solutions (A and B) for constrained optimization problems is to use the following parameterless strategy to select a winner:

1. If A and B are feasible, choose the one having smaller objective (f) value,
2. If A is feasible and B is not, choose A and vice versa,
3. If A and B are infeasible, choose the one having smaller CV value.

Step 5 puts the algorithm into a loop until a termination plan (TP) in Step 13 is satisfied. The loop involves Steps 6 to 12. TP may involve achievement of a pre-specified target objective value (f_T) and will be satisfied when $f(\mathbf{x}_t^*) \leq f_T$. TP may simply involve a pre-specified number of iterations T, thereby setting the termination condition $t \geq T$. Other TPs are also possible and this is one of the five plans that is to be chosen by the user.

Step 6 chooses μ solutions from the set S_t by using a selection plan (SP). SP must carefully analyze S_t and select better solutions (using f and CV) of S_t. A metaheuristic method will vary largely based on what SP is used. The μ solutions form a new set P_t.

In Step 7, solutions from P_t are used to create a set of λ new solutions using a generation plan (GP). This plan provides the main search operation of the metaheuristic algorithm. The plans will be different for function optimization and combinatorial optimization problems. The created solutions are saved in set C_t.

Step 8 chooses ρ worse or random solutions of S_t to be replaced by a replacement plan (RP). The solutions to be replaced are saved in R_t. In some metaheuristic algorithms, R_t can simply be P_t (requiring $\rho = \mu$), thereby replacing the very solutions used to create the new solutions. To make the algorithm more greedy, R_t can be ρ worst members of S_t.

In Step 9, ρ selected solutions from S_t are to be replaced by ρ other solutions. Here, a pool of at most $(\mu + \lambda + \rho)$ solutions of the combined pool $P_t \cup C_t \cup R_t$ is used to choose ρ solutions using an update plan (UP). If $R_t = P_t$, then the combined pool need not have both R_t and P_t in order to avoid duplication of solutions. UP can simply be choosing the best ρ solutions of the pool. Note that a combined pool of $R_t \cup C_t$ or $P_t \cup C_t$ or simply C_t is also allowed. If members of R_t are included in the combined pool, then the metaheuristic algorithm possess an elite-preserving property, which is desired in an efficient optimization algorithm.

RP and UP play an important role in determining the robustness of the metaheuristic. A greedy approach of replacing the worst solutions with the best solutions may lead to faster yet premature convergence. Maintaining diversity in candidate solutions is an essential aspect to be considered when designing RPs and UPs.

Step 10 evaluates each of the new members of S_t and Step 11 updates the set-best member $\mathbf{x}_t^*$ using the best members of updated S_t. When the algorithm satisfies TP, the current-best solution is declared as the near-optimal solution.

The above metaheuristic algorithm can also represent a single solution optimization procedure by setting $N = 1$. This will force $\mu = \rho = 1$. If a single new solution is to be created in Step 7, $\lambda = 1$. In this case, SP and RP are straightforward procedures of choosing the singleton solution in S_t. Thus, $P_t = R_t = S_t$. The generation plan (GP) can choose a solution in the neighborhood of singleton solution in S_t using a Gaussian or other distribution. The update plan (UP) will involve choosing a single solution from two solutions of $R_t \cup C_t$ (elite preservation) or choosing the single new solution from C_t alone and replacing P_t.

By understanding the role of each of the five plans and choosing their sizes appropriately, many different metaheuristic optimization algorithms can be created. All five plans may involve stochastic operations, thereby making the resulting algorithm stochastic.

With this generic framework in mind, we are now ready to describe a number of existing metaheuristic algorithms. Readers are encouraged to draw parallels between different operations used in these algorithms with the five plans discussed above. It should be noted that this may not always be possible since some operations may be spread across or combine multiple plans.

3. Ant Colony Optimization (ACO)

Ant colony optimization (ACO) algorithms fall into the broader area of swarm intelligence in which problem solving algorithms are inspired by the behavior of swarms. Swarm intelligence algorithms work with the concept of self-organization, which relies on any form of central control over the swarm members. A detailed overview of the self-organization principles exploited by these algorithms, as well as examples from biology, can be found in [57]. Here, we specifically discuss the ant colony optimization procedure.

As observed in nature, ants often follow each other on a particular path to and from their nest and a food source. An inquisitive mind might ask the question: 'How do ants determine such a path and importantly, is the path followed by the ants optimal in any way?' The answer lies in the fact that when ants walk, they leave a trail of chemical substance called *pheromone*. The pheromone has the property of depleting with time. Thus, unless further deposit of pheromone takes place on the same pheromone trail, the ants will not be able to follow one another. This can be observed in nature by wiping away a portion of the trail with a finger. The next ant to reach the wiped portion can be observed to stop, turn back or wander away. Eventually, an ant will 'risk' crossing the wiped portion and finds the pheromone trail again. Thus, in the beginning, when the ants have no specific food source, they wander almost randomly on the lookout for one. While they are on the lookout, they deposit a pheromone trail which is followed by other ants if the pheromone content is strong enough. Stronger the pheromone trail, higher is the number of ants that are likely to follow the trail. When a particular trail of ants locates a food source, the act of repeatedly carrying small amounts of food back to the nest increases the pheromone content on that trail, thereby attracting more and more ants to follow it. However, if an ant ends up just wandering and eventually becomes unsuccessful in locating a food source, despite attracting some ants to follow its trail, will only frustrate the following ants and eventually the pheromone deposited on that trail will decrease. This phenomenon is also true for sub-optimal trails, i.e. trails to locations with limited food source.

These ideas can be used to design an optimization algorithm for finding the shortest path. Let us say that two different ants are able to reach a food source using two different pheromone trails – one shorter and the other somewhat longer from source to destination. Since the strength of a pheromone trail is related to the amount of pheromone deposit, for an identical number of ants in each trail the pheromone content will get depleted quickly for the longer trail, and by the same reason the pheromone content on the shorter trail will get increasingly strengthened. Thus, eventually there will be so few ants following the longer trail, that pheromone content will go below a limit that is not enough to attract any further ants. This fact of a strengthening shorter trails with pheromone allows the entire ant colony to discover an eventual minimal trail for a given source-destination combination. Interestingly, such a task is also capable of identifying the shortest distance configuration in a scenario having multiple food source and multiple nests.

Although observed in 1944 [58], the ants' ability to self-organize to solve a problem was only demonstrated in 1989 [59], Dorigo and his coauthors suggested an ant colony optimization method in 1996 [60] that made the approach popular in solving combinatorial optimization problems. Here, we briefly describe two variants of the ant colony optimization procedure – the ant system (AS) and the ant colony system (ACS) – in the context of a traveling salesman problem (TSP).

In a TSP, a salesman has to start from a city and travel to the remaining cities one at a time, visiting each city exactly once so that the overall distanced traveled is minimum. Although sounds simple, such a problem is shown to be NP-hard [61] with increasing number of cities. The first task in using an ACO is to represent a solution in a manner amenable by an ACO. Since pheromone can dictate the presence, absence, and strength of an path, a particular graph connecting cities in a TSP problem is represented in an ACO by the pheromone value of each edge. While at a particular node (i), an ant (say k-th ant) can then choose one of the neighboring edges (say from node i to neighboring node j) based on the existing

pheromone value (τ_{ij}) of each edge (ij edge) probabilistically as follows:

$$p_{ij}^k = \begin{cases} \frac{\tau_{ij}^\alpha \eta_{ij}^\beta}{\sum_{l \in C_k} \tau_{ij}^\alpha \eta_{ij}^\beta}, & \text{if } j \in C_k, \\ 0, & \text{otherwise,} \end{cases} \tag{4}$$

where τ_{ij} is the existing pheromone value of the ij-th edge, heuristic information η_{ij} is usually the part of the objective value associated with ij-th edge, C_k is the set of cities that are not visited by the k-th ant yet and α and β are problem specific parameters indicating relative importance of existing pheromone content and heuristic information in deciding the probability event, respectively. For the ant system (AS) approach, η_{ij} is chosen as the inverse of the distance between nodes i and j. Thus, among all allowable node from C_k, the probability is highest for the node (j) that is nearest to node i.

After the complete path is identified, the pheromone content of each edge is updated as follows:

$$\tau_{ij} = (1-\rho)\tau_{ij} + \rho \sum_{k=1}^{m} \Delta\tau_{ij}^k, \tag{5}$$

where $\rho \in (0,1)$ is the evaporation rate or pheromone decay rate, m is the number of ants in the AS, $\Delta\tau_{ij}^k$ is the quantity of pheromone laid by the k-th ant on edge ij given by:

$$\Delta\tau_{ij}^k = \begin{cases} \frac{Q}{L_k}, & \text{if ant } k \text{ used the edge } ij \text{ in its tour,} \\ 0, & \text{otherwise.} \end{cases} \tag{6}$$

The parameter Q is a constant and L_k is the overall length of the tour by ant k. Thus, if the k-th ant finds a shorter tour (having smaller L_k), the pheromone content τ_{ij}^k gets increased by a larger amount. The first term in Equation (5) causes evaporation or decay of pheromone and the second term strengthens the pheromone content if an overall shorter tour is obtained.

The AS approach works as follows. Initially, random values of pheromone content within a specified range are assumed for each edge. Each ant in the system starts with a random city, and visits allowable neighboring cities selected probabilistically using Equation (4). For m ants, m such trails are constructed. Thereafter, pheromone levels are updated for each edge using Equation (5). The whole process is repeated until either all ants take the same trail or the maximum number of iterations is reached, in which case the shortest path found becomes the solutions. The AS algorithm has been used to solve quadratic assignment problems [62], job-shop scheduling problems [63], vehicle routing problem [64], etc.

In the ant colony system (ACS) approach, in addition to a global pheromone update rule described later, a local pheromone update rule is also applied [65]. Each ant modifies the pheromone content of each edge as soon as it traverses it, as follows:

$$\tau_{ij} = (1-\psi)\tau_{ij} + \psi\tau_0, \tag{7}$$

where $\psi \in (0,1)$ and τ_0 is the initial value of the pheromone. The effect of this local pheromone update is to diversify the search procedure by allowing subsequent ants to be aware of the edges chosen by previous ants. The global pheromone update rule in ACS only effects the best tour of the k-ants in the current iteration (having a tour length L_b). The rest of the edges are only subject to evaporation. That is,

$$\tau_{ij} = \begin{cases} (1-\rho)\tau_{ij} + \rho\Delta\tau_{ij}, & \text{if edge } ij \text{ belongs to tour,} \\ (1-\rho)\tau_{ij}, & \text{otherwise.} \end{cases} \tag{8}$$

where $\Delta\tau_{ij} = 1/L_b$ is used. Another modification adopted in ACS concerns the edge probability. For an ant k, the choice of city j depends on a parameter q_0 (user-defined within [0,1]). If a random number $q \in [0,1]$ is such that $q \leq q_0$, then the next city is

$$j = \text{argmax}_{l \in C_k} \left\{ \tau_{il} \eta_{il}^\beta \right\}, \tag{9}$$

Algorithm 2 Ant Colony Optimization

Require: m, n, α, β, ρ, ψ, q_0, τ_0, Q

Set iteration counter $t = 0$
Set best route length L_{best} to ∞
if AS is desired **then**
  Randomly initialize edge pheromone levels within a range
end if
if ACS is desired **then**
  Initialize all edge pheromone levels to τ_0
end if
repeat
  for $k = 1$ **to** m **do**
    Randomly place ant k at one of the n cities
    repeat
      if ACS is desired **and** $rand(0, 1) \leq q_0$ **then**
        Choose next city to be visited using Equation (9)
      else
        Choose next city to be visited using Equation (4)
      end if
      if ACS is desired **then**
        Update current edge using local pheromone update rule in Equation (7)
      end if
    until unvisited cities exist
    Find best route having length L_b
    if ACS is desired **then**
      Update edges of best route using global pheromone update rule in Equation (8)
    end if
    if AS is desired **then**
      for all edges **do**
        Update current edge using pheromone update rule in Equation (5)
      end for
    end if
  end for
  if $L_b < L_{best}$ **then**
    $L_{best} = L_b$
  end if
  $t \leftarrow t + 1$
until TP is satisfied
Declare route length L_{best} as near-optimal solution

otherwise Equation (4) is used.

Both AS and ACS can be represented by the pseudocode in Algorithm 2.

ACO for solving continuous optimization problems is not straightforward, yet attempts have been made [66, 67, 68] to change the pheromone trail model, which is by definition a discrete probability distribution, to a continuous probability density function. Moreover, presently there is no clear understanding of which algorithms may be called ant based. Stigmergy optimization is the generic name given to algorithms that use multiple agents exchanging information in an indirect manner [69].

4. Artificial Bee Colony Optimization (ABCO)

Karaboga [70, 71] suggested a global-cum-local search based optimization procedure based on the bee's foraging for nectar (food) in a multi-dimensional space. Food sources span the entire variable space and the location of a food source is a point in the variable space. The nectar content at a food source is considered as the objective function of that point in variable space. Thus, internally ABCO method maximizes the objective function similar to bee's locating the food source having the highest nectar content. In other words, the task of finding the optimal solution of an objective function $f(\mathbf{x})$ is converted to an artificial bee colony optimization problem in which artificial bees will wander in an artificial multi-dimensional space to locate the highest producing nectar source.

In order to achieve the search task, the essential concept of bee colony's food foraging procedure is simulated in an artificial computer environment. A population of initial food sources $\mathbf{x}^{(k)}$ ($k = 1, 2, \ldots, N$, where N is the colony size) is randomly created in the entire variable space:

$$x_i^{(k)} = x_i^{(L)} + u_i \left(x_i^{(U)} - x_i^{(L)} \right), \tag{10}$$

where u_i is a random number in $[0, 1]$. Three different types of bees are considered, each having to do a different task. The *employed* bees consider a food source from their respective *memories* and find a new food source $\mathbf{v}_e$ in its neighborhood. Any neighborhood operator can be used for this purpose. Simply, a uniformly distributed food source within $\pm\mathbf{a}$ of the current memory location $\mathbf{x}_m$ can be used:

$$v_{e,i} = x_{m,i} + \phi_i \left(x_{m,i} - x_i^{(k)} \right), \tag{11}$$

where $\mathbf{x}^{(k)}$ is a randomly selected food source and ϕ_i is a random number in $[-a_i, a_i]$. The newly created food source $\mathbf{v}_e$ is then compared with $\mathbf{x}_m$ and the better food source is kept in the memory of the employed bee. The number of employed bees are usually set as 50% of the food sources (N).

Employed bees share their food source information (that are stored in their memories) with *onlooker* bees, who stay in the bee hive and observe the foraging act of employed bees. An onlooker bee chooses the food source location $\mathbf{v}_e$ found by an employed bee with a probability which is proportional to the nectar content of the food source $\mathbf{v}_e$. The higher the nectar content, the higher is the probability of choosing the food source. Once a food source is selected, it is again modified to find $\mathbf{v}_o$ in its neighborhood by using a similar procedure with the selected $\mathbf{v}_e$, as depicted in Equation (11). The better of the two food sources are kept in the memory of the onlooker bee. The number of onlooker bees are usually set as 50% of the food sources (N).

Finally, the third kind of bees – the *scout* bees – randomly choose a food source location using Equation (10) and act more like global overseers. If any employed bee cannot improve their memory location by a predefined number of trials (a limit parameter L), it becomes a scout bee and reinitializes its memory location randomly in the variable space. The number of scout bee is usually chosen as one. The algorithm is run for a maximum of T generations.

Each food source is associated with either an employed or an onlooker bee, thereby having a single food source in each of them. The overall ABCO algorithm has the typical structure as shown in Algorithm 3.

ABCO method has been applied to solve integer programming [72, 73, 74], combinatorial optimization problems [75, 76] and multi-objective optimization problems [77].

Algorithm 3 Artificial Bee Colony Optimization Algorithm

Require: N, L

1: Set iteration counter $t = 0$
2: Initialize $N/2$ employer bees (solutions) to $N/2$ food sources using Equation (10)
3: **repeat**
4: Find nectar content at employer bee locations (i.e. evaluate solutions $\mathbf{x}_m$)
5: Move each employer bee to a new food source in its neighborhood using Equation (11)
6: Find nectar content at new employer bee locations (i.e. evaluate solutions $\mathbf{v}_e$)
7: Record employer bees that return to $\mathbf{x}_m$ due to lower nectar content at $\mathbf{v}_e$
8: Recruit $N/2$ onlooker bees to employer bee locations with probabilities proportional to their nectar content
9: Move each onlooker bee to a new food source in its neighborhood (similar to Equation (11))
10: Find nectar content at new onlooker bee locations (i.e. evaluate solutions $\mathbf{v}_o$)
11: Move each employer bee to the best location found by onlooker bees assigned to it
12: Record the best food source among all $\mathbf{x}_m$, $\mathbf{v}_e$ and $\mathbf{v}_o$
13: Convert employer bees that cannot find better food sources in L trials into scout bees
14: Initialize scout bees, if any, using Equation (10)
15: $t \leftarrow t + 1$
16: **until** TP is satisfied
17: Declare the best food source (near-optimal solution)

5. Artificial Immune System (AIS)

Immune systems in biological organisms protect living cells from an attack from foreign bodies. When attacked by *antigens*, different antibodies are produced to defend the organism. In a macro sense, the natural immune system works as follows. Antibodies that are efficient in fighting antigens are cloned, hyper-mutated and more such antibodies are produced to block the action of antigens. Once successful, the organism learns to create efficient antibodies. Thus, later if similar antigens attack, the organism can quickly produce the right antibodies to mitigate their effect. The process of developing efficient antibodies can be simulated to solve optimization problems, in which antibodies can represent decision variables and the process of cloning, hyper-mutation and selection can be thought as a search process for finding *optimal* antibodies that tend to match or reach the extremum values of antigens representing objectives. There are mainly two different ways such an immunization task is simulated, which we discuss next.

5.1. Clonal Selection Algorithm

Proposed by Burnet [78] in 1959, the clonal selection procedure works with the concept of cloning and *affinity maturation*. When exposed to antigens, B-cells (so called because they are produced in the bone marrow) that best bind with the attacking antigens get emphasized through cloning. The ability of binding depends on how well the *paratope* of an antibody matches with the *epitope* of an antigen. The closer the match, the stronger is the affinity of binding. In the context of a minimization problem, the following analogy can drive the search towards the minimum. The smaller the value of the objective function for an antibody representing a solution, the higher can be the affinity level. The cloned antibodies can then undergo hyper-mutation according to an inverse relationship to their affinity level. Receptor editing and introduction of random antibodies can also be implemented to maintain a diverse set of antibodies (solutions) in the population. These operations sustain the affinity maturation process.

The clonal selection algorithm CLONALG [79, 80] works as follows. First, a population of N antibodies is created at random. An antibody can be a vector of real, discrete or Boolean variables, or it can be a permutation or a combination of the above, depending on the optimization problem being solved. At every iteration, a small fraction (say, n out of N) of antibodies is selected from the population

based on their affinity level. These high-performing antibodies are then cloned and then hyper-mutated to construct N_c new antibodies. The cloning is directly proportional to the ranking of the antibodies in terms of their affinity level. The antibody with the highest affinity level in a population has the largest pool of clones. Each clone antibody is then mutated with a probability that is higher for antibodies having a lower affinity level. This allows a larger mutated change to occur in antibodies that are worse in terms of antigen affinity (or worse in objective values). Since a large emphasis of affinity level is placed on the creation mechanism of new antibodies, some randomly created antibodies are also added to the population to maintain adequate diversity. After all newly created antibodies are evaluated to obtain their affinity level, $R\%$ worst antibodies are replaced by the best newly created antibodies. This process continues till a pre-specified termination criterion is satisfied. Algorithm 4 shows the basic steps.

Algorithm 4 Artificial Immune Systems - Clonal Selection Algorithm

Require: N, n, N_c, R
1: Set iteration counter $t = 0$
2: Randomly initialize N antibodies to form S_t
3: **repeat**
4: Evaluate affinity (fitness) of all antibodies in S_t
5: Select n antibodies with highest affinities to form P_t
6: Clone the n antibodies in proportion to their affinities to generate N_c antibodies
7: Hyper-mutate the N_c antibodies to form C_t
8: Evaluate affinity of all antibodies in C_t
9: Replace $R\%$ worst antibodies in S_t with the best antibodies from C_t
10: Add new random antibodies to S_t
11: $t \leftarrow t + 1$
12: **until** TP is satisfied
13: Declare population best as near-optimal solution

On careful observation, the clonal selection algorithm described above is similar to a genetic algorithm (GA), except that no recombination-like operator is used in the clonal selection algorithm. The clonal selection algorithm is a population-based optimization algorithm that uses overlapping populations and heavily relies on a mutation operator whose strength directly relates to the value of objective function of the solution being mutated.

5.2. Immune Network Algorithm

A natural immune system, although becomes active when attacked by external antigen, sometimes can work against its own cells, thus stimulating, activating or suppressing each other. To reduce the chance of events, the clonal selection algorithm has been modified to recognize antibodies (solutions) that are similar to one another in an evolving antibody population. Once identified, similar antibodies (in terms of a distance measure) are eliminated from the population through *clonal suppression*. Moreover, antibodies having very small affinity level towards attacking antigens (meaning worse than a threshold objective function value) are also eliminated from the population. In the artificial immune network (aiNET [81]) algorithm, a given number of affinity clones are selected to form an immune network. These clones are thought to form a *clonal memory*. The clonal suppression operation is performed on the antibodies of the clonal memory set. The aiNET algorithm works as follows. A population of N antibodies are created at random. A set of n antibodies are selected using their affinity level. Thereafter, each is cloned based on their affinity level in a manner similar to that in the clonal selection algorithm. Each clone is then hyper-mutated as before and are saved within a clonal memory set (M). After evaluation, all memory clones that have objective value worse than a given threshold ϵ are eliminated. Antibodies that are similar (e.g. solutions clustered together) to each other are also eliminated through the clonal suppression process using a suppression threshold σ_s. A few random antibodies are injected in the antibody population as

before, and the process is continued till a termination criterion is satisfied. The pseudocode for a typical immune network algorithm is shown in Algorithm 5.

Algorithm 5 Artificial Immune Systems - Immune Network Algorithm

Require: N, n, N_c, ϵ, σ_s, R
1: Set iteration counter $t = 0$
2: Set clonal memory set $M = \Phi$
3: Randomly initialize N antibodies to form S_t
4: **repeat**
5: Evaluate affinity (fitness) of all antibodies in S_t
6: Select n antibodies with highest affinities to form P_t
7: Clone the n antibodies in proportion to their affinities to generate N_c antibodies
8: Mutate the N_c antibodies in inverse-proportion to their affinities to form C_t
9: Evaluate affinity of all antibodies in C_t
10: $M = M \cup C_t$
11: Eliminate memory clones from M with affinities $< \epsilon$
12: Calculate similarity measure s for all clonal pairs in M
13: Eliminate memory clones from M with $s < \sigma_s$
14: Replace $R\%$ worst antibodies in S_t with the best antibodies from M
15: Add new random antibodies to S_t
16: $t \leftarrow t + 1$
17: **until** TP is satisfied
18: Declare the best antibody in M as near-optimal solution

In the context of optimization, artificial immune system (AIS) algorithms are found to be useful in following problems. Dynamic optimization problems in which the optimum changes with time is a potential application domain [82] simply because AIS is capable of quickly discovering a previously-found optimum if it appears again at a later time. The immune network algorithm removes similar solutions from the population, thereby allowing multiple disparate solutions to co-exist in the population. This allowed AIS algorithms to solve multi-modal optimization problems [83] in which the goal is to find multiple local or global optima in a single run. AIS methods have also been successfully used for solving traveling salesman problems [84]. Several hybrid AIS methods coupled with other evolutionary approaches are proposed [85]. AIS has also been hybridized with ant colony optimization (ACO) algorithms [86]. More details can be found elsewhere [87].

6. Differential Evolution (DE)

Differential evolution was proposed by Storn and Price [88] as a robust and easily parallelizable alternative to global optimization algorithms. DE borrows the idea of self-organizing from Nelder-Mead's simplex search algorithm [89], which is also a popular heuristic search method. Like other population-based metaheuristics, DE also starts with a population of randomly initialized solution vectors. However, instead of using variation operators with predetermined probability distributions, DE uses the difference vector of two randomly chosen members to modify an existing solution in the population. The difference vector is weighted with a user-defined parameter $F > 0$ and to a third (and different) randomly chosen vector as shown below:

$$\mathbf{v}_{i,t+1} = \mathbf{x}_{r_1,t} + F \cdot (\mathbf{x}_{r_2,t} - \mathbf{x}_{r_3,t}). \tag{12}$$

At each generation t, it is ensured that r_1, r_2 and r_3 are different from each other and also from i. The resultant vector $\mathbf{v}_i$ is called a mutant vector of $\mathbf{x}_i$. F remains constant with generations and is fixed in $[0, 2]$ (recommended $[0.4, 1]$ [90]).

DE also uses a variation of crossover in which instead of recombining different members of the population, each member recombines with its own mutant vector. This recombination is similar to the discrete recombination of ES, where each component (j) of the offspring is randomly chosen from one of the parents. Of course, in DE the parents are simply $\mathbf{x}_i$ and its corresponding mutant vector $\mathbf{v}_i$. The recombinant is called a trial vector and denoted by $\mathbf{u}_i$. The crossover can be given by:

$$u_{ji,t+1} = \begin{cases} v_{ji,t+1} & \text{if } (rand_j[0,1] \leq CR \text{ or } j = rand_i) \\ x_{ji,t} & \text{otherwise} \end{cases} \tag{13}$$

where $rand_i$ is a dimension randomly selected once for each i. Thus, $j = rand_i$ ensures that the first condition is true at least once, i.e. at least one of the components of the trial vector $\mathbf{u}_i$ comes from $\mathbf{v}_i$. $CR \in [0, 1]$ is called the crossover constant.

Different DE strategies are generally represented using the notation $DE/x/y/z$ where

1. x specifies the vector to be mutated,
2. y specifies the number of difference vectors to be used,
3. and z denotes the crossover scheme. The one described above is called independent binomial and is represented by "bin".

The most popular variants are $DE/rand/1/bin$ described above and $DE/best/2/bin$ [91] which uses the mutation,

$$\mathbf{v}_{i,t+1} = \mathbf{x}_{best,t} + F \cdot (\mathbf{x}_{r_1,t} - \mathbf{x}_{r_2,t}) + F \cdot (\mathbf{x}_{r_3,t} - \mathbf{x}_{r_4,t}). \tag{14}$$

However, as many as ten different variants were proposed by Price et al. [92] by combining binomial ("bin") and exponential ("exp") crossovers with $DE/rand/1/$, $DE/rand/2/$, $DE/best/1/$, $DE/best/2/$ and $DE/target-to-best/2$. Other variants which use objective function information $DE/rand/2/dir$ [93], trigonometric mutation [94], arithmetic recombination [95] and pure mutants and pure recombinants [92] have also been proposed. A summary of these and many other DE variants can be found in [90]. Here we lay out the pseudocode for the classic $DE/rand/1/bin$ variant in Algorithm 6.

7. Evolution Strategies (ES)

Evolution strategies initially employed the most basic form of adaptation among all of evolutionary algorithms. Developed in the 1960s [96], the simplistic version called the (1+1)-ES uses one parent to produce an offspring through binomially distributed mutation. The better of the two is selected for the next iteration. An approximate analysis of the (1+1)-ES using Gaussian mutation was presented by Rechenberg in [97]. Over the years, much theory in metaheuristics has been developed with regard to convergence of the (1+1)-ES on different problem types. The first multi-membered ES [98], called the (μ+1)-ES or steady state ES uses a population of μ parents and introduced recombination of parents. Two randomly selected parents are crossed over to produce an offspring and after it undergoes mutation, the worst of the μ+1 individuals is eliminated for the next iteration. (μ+1)-ES showed the effectiveness of recombination in speeding up evolution [99]. Schwefel introduced two new versions of the multi-membered ES [100, 101] which generate $\lambda \geq 1$ offspring in each generation. The first version, denoted as $(\mu + \lambda)$-ES, eliminates λ worst individuals from the combined parent-offspring population of $\mu + \lambda$ individuals, thus performing elitism. The second version, denoted as (μ, λ)-ES, simply replaces the μ parents with μ individuals from λ offspring, thus requiring $\lambda > \mu$. Note that $\lambda = \mu$ will not work since all offspring will have to be selected, thus providing no selection pressure. The two versions are also sometimes referred to as plus-selection and comma-selection ES respectively. It is generally recommended that plus-selection be used with discrete search spaces and comma-selection be used for unbounded search spaces [99].

The canonical form of state-of-the-art ES algorithms is $(\mu/\rho \stackrel{+}{,} \lambda)$-ES. Here ρ (mixing number) represents the number of parents (out of μ) that are used for recombination to produce one offspring. Naturally, an ES with $\rho = 1$ uses no recombination. A distinguishing feature of ES is the use of strategy parameters

Algorithm 6 Differential Evolution

Require: N, F, CR

Set iteration counter $t = 0$
Randomly initialize N population members to form S_t
repeat
  Evaluate fitness of all members in S_t
  for $i = 1$ **to** N **do**
    Generate random integer $rand_i$ between 1 and search space dimension D
    for $j = 1$ **to** D **do**
      Generate random number $rand_j \in [0, 1]$
      if $rand_j \leq CR$ **or** $j = rand_i$ **then**
        $u_{ji,t+1} = v_{ji,t+1} = x_{jr_1,t} + F \cdot (x_{jr_2,t} - x_{jr_3,t})$
      else
        $u_{ji,t+1} = x_{ji,t}$
      end if
    end for
    Evaluate trial vector $\mathbf{u}_{i,t+1}$
    if $f(\mathbf{u}_{i,t+1}) \leq f(\mathbf{x}_{i,t})$ **then**
      Replace $\mathbf{x}_{i,t}$ with $\mathbf{u}_{i,t+1}$
    end if
  end for
  $t \leftarrow t + 1$
until TP is satisfied
Declare population best as near-optimal solution

s that are unique for each individual. These so called *endogenous* strategy parameters can evolve with generations. Most self-adaptive ES use these evolvable parameters to tune the mutation strength. These strategy parameters can also be recombined and mutated like the solution vectors. On the other hand, *exogenous* strategy parameters μ, ρ and λ remain unchanged with generations. Algorithm 7 shows the generic ES procedure.

Recombination of the ρ members can either be discrete (dominant) or intermediate. In the former, each dimension of the offspring is randomly chosen from the corresponding dimension of the parents, whereas in the latter, all dimensions of the offspring are arithmetically averaged values of corresponding dimensions over all parents. For strategy parameters intermediate (μ/μ) recombination has been recommended in [99] and [102] to avoid over-adaptation or large fluctuations.

Traditionally, ES variants have only used mutation as the variation operator. As a result, many studies exist which discuss, both theoretically [103, 104] and empirically, the effect of mutation strength on the convergence of ES and the self-adaptation of strategy parameters in mutation operators [105], most notably that of the standard deviation in Gaussian distribution mutation. Self-adaptive ES have emerged to be one of the most competitive metaheuristic methods in recent time. We describe them next.

7.1. Self-adaptive ES

A self-adaptive ES is one in which the endogenous strategy parameters (whether belonging to recombination or mutation) can tune (adapt) themselves based on the current distribution of solutions in the search space. The basic idea is to try bigger steps if past steps have been successful and smaller steps when they have been unsuccessful. However, as mentioned before, most self-adaptive ES are concerned with the strategy parameter of mutation and this is what we discuss here. Discussions on other strategy parameters can be found in [106].

Algorithm 7 $(\mu/\rho \dot{+} \lambda)$ Evolution Strategy

Require: μ, ρ, λ

Set iteration counter $t = 0$
Randomly initialize μ population members in S_t and their strategy parameters $\mathbf{s_t}$
repeat
  Evaluate fitness of all members in S_t
  Set $C_t = \Phi$
  for $i = 1$ **to** λ **do**
    Randomly select ρ members from S_t to form P_t
    Recombine the strategy parameters of the ρ members to get $\mathbf{s'}_t^{(i)}$
    Recombine the members in P_t to form an offspring $\mathbf{x'}_t^{(i)}$
    Mutate the strategy parameters $\mathbf{s'}_t^{(i)}$ to get $\mathbf{s}_{t+1}^{(i)}$
    Mutate offspring $\mathbf{x'}_t^{(i)}$ using $\mathbf{s}_t^{(i)}$ to get $\mathbf{x}_{t+1}^{(i)}$
    $C_t = C_t \cup \mathbf{x}_{t+1}^{(i)}$
  end for
  Evaluate fitness of all members in C_t
  if comma-selection is desired **then**
    Select best μ members from C_t and update S_t
  end if
  if plus-selection is desired **then**
    Select best μ members from $P_t \cup C_t$ and update S_t
  end if
  $t \leftarrow t + 1$
until TP is satisfied
Declare population best as near-optimal solution

For unconstrained real-valued search spaces, the maximum entropy principle says that the variation operator should use a normal distribution so as to not introduce any bias during exploration [99]. The mutation operation on a solution (or 'recombinant' if crossover is used) $\mathbf{x}'$ is therefore given by,

$$\mathbf{x} = \mathbf{x}' + \sigma\mathbf{N}(\mathbf{0}, \mathbf{I}) = \sigma\mathbf{N}(\mathbf{x}', \mathbf{I}), \tag{15}$$

where σ (= $\mathbf{s}$ in Algorithm 7) is the only strategy parameter and $\mathbf{N}(\mathbf{0}, \mathbf{I})$ is a zero-mean unit-variance normally distributed random vector. Thus, each component i of the mutant $\mathbf{x}$ obeys the following density function:

$$prob(x_i') = \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{(x_i' - x_i)^2}{2\sigma^2}\right). \tag{16}$$

The mutation strength σ is thus the standard deviation of the mutant distribution, which implies $\sigma > 0$. It is also sometimes called the step-size when referring to Equation (15). Note that for σ being an endogenous strategy parameter, also undergoes recombination and mutation. As discussed above, recombination of strategy parameters is mainly intended to reduce fluctuations in values and therefore intermediate recombination (arithmetic averaging of standard deviations of parents) is often used. Mutation of the mutation strength is multiplicative since an additive mutation (like that in Equation (15)) cannot guarantee $\sigma > 0$. For a strategy parameter value (or 'recombinant' if crossover is used) σ', the 'log-normal' mutation operator [100] is given by,

$$\sigma = \sigma' \exp(\tau N(0, 1)), \tag{17}$$

where τ is called the learning parameter and is proportional to $1/\sqrt{N}$ (usually $\tau = 1/\sqrt{2N}$), N being the search space dimension. A self-adaptive ES using Equation (17) in Step 10 and Equation (15) in Step 11 of Algorithm 7 is called $(\mu/\rho \overset{+}{,} \lambda)$-$\sigma$SA-ES in literature.

When multiple strategy parameters are used i.e. when $\mathbf{s} = \boldsymbol{\sigma}$, Equation (15) becomes,

$$\mathbf{x} = \mathbf{x}' + \text{diag}(\sigma_1, \sigma_2, \ldots, \sigma_N)\mathbf{N}(\mathbf{0}, \mathbf{I}). \tag{18}$$

Here, diag() is a diagonal matrix with elements σ_1, σ_2, etc. representing the standard deviations for mutation along different dimensions. This allows for axes-parallel ellipsoidal mutant distributions. Self-adaptation in this case is given by the extended log-normal rule [101],

$$\boldsymbol{\sigma} = \exp(\tau_0 N(0, 1)) \begin{pmatrix} \sigma_1' \exp(\tau N_1(0, 1)) \\ \sigma_2' \exp(\tau N_1(0, 1)) \\ \vdots \\ \sigma_N' \exp(\tau N_N(0, 1)) \end{pmatrix}. \tag{19}$$

τ_0 acts as a global learning parameter and τ acts on coordinate-wise on each normal distribution $N_i(0, 1)$. Recommended values for them are $\tau_0 \propto 1/\sqrt{2N}$ and $\tau \propto 1/\sqrt{2\sqrt{N}}$.

7.2. Covariance Matrix Adaptation-ES (CMA-ES)

CMA-ES [107] and its variants [108, 109, 110] represent the state-of-the-art of self-adaptive ES algorithms for continuous search space optimization. The main difference in CMA-ES from self-adaptive ES is that it uses a generic multivariate normal distribution for mutation which is given by

$$\mathbf{x} = \mathbf{x_m} + \sigma\mathbf{N}(\mathbf{0}, \mathbf{C}) = \sigma\mathbf{N}(\mathbf{x_m}, \mathbf{C}), \tag{20}$$

where $\mathbf{C}$ is the covariance matrix which enables arbitrarily rotated ellipsoidal mutant distributions. By contrast, the mutant distribution obtained from Equation (15) is always spherical while that obtained from Equation (18) is always an axes-parallel ellipsoid.

Another important difference is that while self-adaptive ES starts with a population of randomly initialized solutions, CMA-ES starts with a random population mean $\mathbf{x}_m$ and generates λ offspring using Equation (20). $\mathbf{C}$ is initialized to an identity matrix so that the initial offspring distribution is spherical. After evaluating the λ offspring, μ best solutions are selected and recombined using a weighted intermediate recombination given by,

$$\mathbf{x}'_{\mathbf{m}} = \sum_{i=1}^{\mu} w_i \mathbf{x}_{i:\lambda} \tag{21}$$

to produce a single recombinant representing the new population mean $\mathbf{x}'_{\mathbf{m}}$. The best offspring gets the largest weight and the μ-th best offspring gets the least weight. Additionally, the weights satisfy $w_1 \geq w_2 \geq w_3 \ldots \geq w_\mu > 0$, $\sum w_i = 1$ and $\mu_w = 1/\sum w_i^2 \approx \lambda/4$. Due to this selection scheme CMA-ES is also known as $(\mu/\mu_w, \lambda)$-CMA-ES.

Rewriting $\mathbf{x}'_{\mathbf{m}}$ as below, we can define $\mathbf{y}$ and $\mathbf{y}_{\mathbf{w}}$.

$$\mathbf{x}'_{\mathbf{m}} = \mathbf{x}_{\mathbf{m}} + \sigma \sum_{i=1}^{\mu} w_i \left(\frac{\mathbf{x}_{i:\lambda} - \mathbf{x}_{\mathbf{m}}}{\sigma} \right) = \mathbf{x}_{\mathbf{m}} + \sigma \sum_{i=1}^{\mu} w_i \mathbf{y}_{i:\lambda} = \mathbf{x}_{\mathbf{m}} + \sigma \mathbf{y}_{\mathbf{w}}. \tag{22}$$

Next the step-size σ and the covariance matrix $\mathbf{C}$ are updated using the following formulae:

$$\mathbf{p}'_\sigma = (1 - c_\sigma)\mathbf{p}_\sigma + \sqrt{1 - (1 - c_\sigma)^2}\sqrt{\mu_w}\sqrt{\mathbf{C}^{-1}}\mathbf{y}_{\mathbf{w}}, \tag{23}$$

$$\mathbf{p}'_{\mathbf{c}} = (1 - c_c)\mathbf{p}_{\mathbf{c}} + \mathbf{1}_{(||\mathbf{p}'_\sigma|| < 1.5\sqrt{N})}\sqrt{1 - (1 - c_c)^2}\sqrt{\mu_w}\mathbf{y}_{\mathbf{w}}, \tag{24}$$

$$\mathbf{C}' = (1 - c_1 - c_\mu)\mathbf{C} + c_1 \mathbf{p}'_{\mathbf{c}} {\mathbf{p}'_{\mathbf{c}}}^T + c_\mu \sum_{i=1}^{\mu} w_i \mathbf{y}_{i:\lambda} \mathbf{y}_{i:\lambda}^T \tag{25}$$

$$\sigma' = \sigma \exp\left(\frac{c_\sigma}{d_\sigma} \left(\frac{||\mathbf{p}'_\sigma||}{E\left[||\mathbf{N}(\mathbf{0}, \mathbf{I})||\right]} - 1 \right) \right) \tag{26}$$

Here, the values $c_c, c_\sigma \approx 4/N$, $c_1 \approx 2/N^2$, $c_\mu \approx \mu_w/N^2$ and $d_\sigma = 1 + \sqrt{\mu_w/N}$ are often used. $\mathbf{1}_{(||\mathbf{p}'_\sigma|| < 1.5\sqrt{N})}$ represents an indicator function which is equal to one when its subscript is true, else it is equal to zero. $E\left[||\mathbf{N}(\mathbf{0}, \mathbf{I})||\right]$ represents the expected norm of a normally distributed random vector. $\mathbf{p}_{\mathbf{c}}$ and $\mathbf{p}_\sigma$ are called the evolution paths of the covariance matrix and step-size, respectively. Initializations $\mathbf{C} = \mathbf{I}$ and $\mathbf{p}_{\mathbf{c}}, \mathbf{p}_\sigma = \mathbf{0}$ are used at the start of the algorithm. The above updates ensure that $\mathbf{C}$ is symmetric and positive semi-definite, as required for covariance matrices, at each iteration. The derivation of the above equations is mathematically involved and readers are referred to the original CMA-ES paper [111] and [107, 108, 109] for details and explanation of parameter settings. The pseudocode for CMA-ES is shown in Algorithm 8 for an N dimensional search space.

8. Evolutionary Programming (EP)

Evolutionary programming together with genetic algorithms and evolutionary strategies are the earliest paradigms of evolutionary algorithms. It was developed by Lawrence J. Fogel [112] in 1960. The main difference from GA and ES is that EP does not use a recombination or crossover operator. The reason for this is that EP was originally designed as an abstraction of evolution at the macro level of species, rather than at the genetic level. Thus instead of individuals of a population which can mate and reproduce, EP considers each individual as a reproducing population and the aim is to mimic the linkage between parent and its offspring [113, 114].

EP, like any other evolutionary algorithm, starts with a population of μ randomly initialized population members. Each member undergoes mutation to produce an offspring. Different EP variants use different mutation schemes. The original EP [112] described the evolution of finite state machines using uniform random mutations on discrete variables. David B. Fogel [115] introduced normally distributed

Algorithm 8 $(\mu/\mu_w, \lambda)$ Covariance Matrix Adaptation - Evolution Strategy

Require: λ, μ ($= \lambda/2$ typically)
Set recombination weights $w_i \; \forall \; i = 1, \ldots, \mu$, Recommended $w_i = \ln(\frac{\lambda+1}{2}) - \ln(i)$
Renormalize weights such that $\sum w_i = 1$
Set $\mu_w = 1/\sum w_i^2$, $c_c, c_\sigma \approx 4/N$, $c_1 \approx 2/N^2$, $c_\mu \approx \mu_w/N^2$ and $d_\sigma = 1 + \sqrt{\mu_w/N}$
Set iteration counter $t = 0$
Set $\mathbf{C}_t = \mathbf{I}$, $\mathbf{p}_{\mathbf{c}t}, \mathbf{p}_{\sigma t} = \mathbf{0}$
Randomly initialize mean point $\mathbf{x}_{\mathbf{m}t}$ and step-size $\sigma_t > 0$, Recommended $\sigma_t = 0.3$
repeat
  Generate λ offspring from $\mathbf{x}_{\mathbf{m}t}$ using Equation (20) and form C_t
  Evaluate fitness of $f(\mathbf{x})$ all members in C_t
  Sort members of C_t according to their fitness
  Choose μ best individuals from C_t and form P_t
  Generate updated mean $\mathbf{x}'_{\mathbf{m}}$ from P_t using Equation (21)
  Generate updated step-size evolution path $\mathbf{p}'_\sigma$ using Equation (23)
  Generate updated covariance evolution path $\mathbf{p}'_{\mathbf{c}}$ using Equation (24)
  Generate updated covariance matrix $\mathbf{C}'$ using Equation (25)
  Generate updated step-size σ' using Equation (26)
  $\mathbf{x}_{\mathbf{m}t+1} = \mathbf{x}'_{\mathbf{m}}$
  $\mathbf{p}_{\mathbf{c}t+1} = \mathbf{p}'_{\mathbf{c}}$ and $\mathbf{p}_{\sigma t+1} = \mathbf{p}'_\sigma$
  $\mathbf{C}_{t+1} = \mathbf{C}'$ and $\sigma_{t+1} = \sigma'$
  $t \leftarrow t + 1$
until TP is satisfied
Declare current $\mathbf{x}_{\mathbf{m}}$ as near-optimal solution

mutations for real-valued variables where the standard deviation for an individual's mutation is a function of its fitness value, i.e.,

$$\begin{aligned} \mathbf{x} &= \mathbf{x}' + \sigma\mathbf{N}(\mathbf{0}, \mathbf{I}), \\ \sigma &= \sqrt{G(f(\mathbf{x}'), \kappa)}, \end{aligned} \tag{27}$$

where G is a fitness function that scales the objective function values to positive values and also sometimes uses random alteration κ. Fogel later introduced meta-EP [116] that uses self-adaptation in a manner somewhat similar to that used in ES for multiple strategy parameters, i.e.,

$$\begin{aligned} \mathbf{x} &= \mathbf{x}' + \text{diag}(\sigma_1, \sigma_2, \ldots, \sigma_N)\mathbf{N}(\mathbf{0}, \mathbf{I}), \\ \sigma_i^2 &= {\sigma'_i}^2 + \sqrt{\zeta}\sigma'_i N_i(0, 1) \; \forall i = 1, \ldots, N. \end{aligned} \tag{28}$$

where ζ is an exogenous parameter. Meta-EP self-adaptation differs with respect to the underlying stochastic process. Unlike the log-normal mutation in ES which guarantees positiveness of σ, negative variances can occur in EP. When this happens $\sigma_i = \sqrt{\epsilon_c} > 0$ is used. Later versions of EP [117] use ES's self-adaptation as shown in Equation (19). Fogel also proposed Rmeta-EP [118] for $N = 2$ dimensions which uses correlation coefficients similar to CMA-ES. However, its generalization for $N > 2$ is not obvious [119].

Once μ offspring are created using mutation, EP uses a kind of $(\mu + \mu)$ stochastic selection. For each solution in the combined parent-offspring population of size 2μ, q other solutions are randomly selected from the population. The score $w \in \{0, \ldots, q\}$ of each solution is the number of those q solutions that are worser than it, i.e.,

$$w_i = \sum_{j=1}^{q} \begin{cases} 1 & \text{if } f(\mathbf{x}_i) \leq f(\mathbf{x}_{j:2\mu}) \\ 0 & \text{otherwise} \end{cases} \; \forall i = \{1, \ldots, 2\mu\}. \tag{29}$$

Thereafter, μ individuals with the best scores are chosen for the next iteration. Note that this selection strategy becomes more and more deterministic, approaching that in ES, as q increases. A little thought also reveals that the selection is elitist. The pseudocode for EP is shown in Algorithm 9[1].

Algorithm 9 Evolutionary Programming

Require: μ, q, Mutation parameters ($\{G, \kappa\}$ **or** $\{\zeta, \epsilon_c\}$ **or** $\{\tau_0, \tau\}$)
Set iteration counter $t = 0$
Randomly initialize μ population members and σ_i $\forall i = \{1, \ldots, N\}$ (if used) to form S_t
repeat
  Evaluate fitness $f(\mathbf{x}')$ of all members in $P_t = S_t$
  if standard EP is used **then**
    Mutate individuals in P_t using Equation (27) to form C_t
  end if
  if meta-EP is used **then**
    Mutate individuals in P_t using Equation (28) to form C_t
  end if
  if EP with ES's self-adaptation is used **then**
    Mutate individuals in P_t using Equations (18) and Equation (19) to form C_t
  end if
  for $i = 1$ **to** 2μ **do**
    Select $\mathbf{x}_i$ from $P_t \cup C_t$
    Set $w_i = 0$
    Select q individuals randomly from $P_t \cup C_t$ to form Q_t
    for $j = 1$ **to** q **do**
      Select $\mathbf{x}_j$ from Q_t
      if $f(\mathbf{x}_i) \leq f(\mathbf{x}_j)$ **then**
        $w_i \leftarrow w_i + 1$
      end if
    end for
  end for
  Sort members of $P_t \cup C_t$ according to their scores w
  Replace S_t with μ best score members of $P_t \cup C_t$
  $t \leftarrow t + 1$
until TP is satisfied
Declare population best as near-optimal solution

9. Genetic Algorithms (GA)

Genetic algorithms are among the most popular of metaheuristic techniques. Introduced in 1975 through the seminal work of John Holland [120], its framework for search and optimization, now popularly known as Binary-coded Genetic Algorithm (BGA), was developed by Goldberg [121]. In a BGA, solutions are represented using binary-bit strings which are iteratively modified through recombination and random bit-flips, much like the evolution of genes in nature, which also undergo natural selection, genetic crossovers and mutations. The ease of implementation and generality of BGA made it a popular choice of metaheuristic in the following years. However, inherent problems were also identified along the

[1]Note that this section used 'primed' symbols, $\mathbf{x}'$ and σ_i', even though they are not recombinants, in order to facilitate comparison with ES

way, most notably Hamming cliffs, unnecessary search space discretization and ineffective schema propagation in case of more than two virtual alphabets. These paved the way for Real-parameter Genetic Algorithm (RGA). In an RGA, a solution is directly represented as a vector of real-parameter decision variables. Starting with a population of such solutions (usually randomly created), a set of genetic operations (such as selection, recombination, mutation, and elite-preservation) are performed to create a new population in an iterative manner. Clearly, the selection plan (SP) must implement the selection operator of RGAs, the generational plan (GP) must implement all variation operators (such as recombination and mutation operators), and the update plan (UP) must implement any elite-preservation operator. Although most RGAs differ from each other mainly in terms of their recombination and mutation operators, they mostly follow one of the two algorithmic models discussed later. First, we shall discuss some commonly-used GA operators which can be explained easily with the help of the five plans outlined before.

Among GA's many selection operators, the proportionate selection scheme (or roulette wheel selection), the ranking scheme and the tournament selection operators are most popular. With the proposed metaheuristic algorithm, these operators must be represented in the selection plan. For example, the proportionate selection operator can be implemented by choosing the i-th solution for its inclusion to P_t with a probability $f_i / \sum_i f_i$. This probability distribution around S_t is used to choose μ members of P_t. The ranking selection operator uses *sorted ranks* instead of actual function values to obtain probabilities of solutions to be selected. The tournament selection operator with a size s can be implemented by using a SP in which s solutions are chosen and the best is placed in P_t. The above procedure needs to be repeated μ times to create the parent population P_t.

Traditionally, BGAs have used variants of point crossover and bit-flip mutation operators. RGAs, on the other hand, use blending operators [122] for recombination of solutions. BLX-α [123], simulated binary crossover (SBX) [124] and fuzzy recombination [125] are often used in literature. SBX emulates a single-point binary crossover in that the average decoded values of parents and their offspring are equal. The offspring are thus symmetrically located with respect to the parents. Moreover, it uses a polynomial probability distribution function which ensures that the probability of creating offspring closer to the parents is greater. The distribution can also be easily modified when the variables are bounded [124]. Mutation operators for RGAs such as random mutation [126], uniform mutation [127], non-uniform mutation [126], normally distributed mutation [128] and polynomial mutation [129] have also been developed.

Elite preservation in a GA is an important task [130]. This is enforced by allowing best of parent and offspring populations to be propagated in two consecutive iterations. The update plan of the proposed algorithm-generator achieves elite preservation in a simple way. As long as the previous population S_t or the chosen parent population P_t is included in the update plan for choosing better solutions, elite preservation is guaranteed. Most GAs achieve this by choosing the best μ solutions from a combined population of P_t and C_t.

A niche-preservation operator is often used to maintain a diverse set of solutions in the population. Only the selection plan gets affected for implementing a niche-preservation operator. While choosing the parent population P_t, care should be taken to lower the selection probability of population-best solutions in P_t. Solutions with a wider diversity in their decision variables must be given priorities. The standard sharing function approach [131], clearing approaches [132], and others can be designed using an appropriate selection plan.

A mating restriction operator, on the other hand, is used to reduce the chance of creating *lethal* solutions arising from mating of two dissimilar yet good solutions. This requires a SP in which the procedures of choosing of each of the μ parents become dependent on each other. Once the first parent is chosen, the procedure of choosing other parents must consider a similarity measure with respect to the first parent.

9.1. Generational Versus Steady-State GAs

Evolutionary algorithms, particularly genetic algorithms, are often used with a generational or with a steady-state concept. In the case of former, a complete population of λ solutions are first created before making any further decision. The proposed metaheuristic algorithm can be used to develop a generational GA by repeatedly using the SP-GP plans to create λ new offspring solutions. Thereafter, the replacement plan simply chooses the whole parent population to be replaced, or $R_t = S_t$ (and $\rho = \mu$). With an elite-preservation operator, the update plan chooses the best μ solutions from the combined population $S_t \cup C_t$. In a GA without elitism, the update plan only chooses the complete offspring population C_t.

On the other extreme, a steady-state GA can be designed by using a complete SP-GP-RP-UP cycle for creating and replacing only one solution ($\lambda = r = 1$) in each iteration. It is interesting to note that the SP can use a multi-parent ($\mu > 1$) population, but the GP creates only one offspring solution from it. We shall return to two specific steady-state evolutionary algorithms later. The generational gap GAs can be designed with a non-singleton C_t and R_t (or having $\lambda > 1$ and $r > 1$).

The generational model is a direct extension of the canonical binary GAs to real-parameter optimization. In each iteration of this model, a complete set of N new offspring solutions are created. For preserving elite solutions, both the parent and offspring populations are compared and the best N solutions are retained. In most such generational models, the tournament selection (SP) is used to choose two parent solutions and a recombination and a mutation operator are applied to the parent solutions to create two offspring solutions. Algorithm 10 shows the pseudocode for the above described GA.

Algorithm 10 Genetic Algorithm

Require: N, p_c, p_m
1: Set iteration counter $t = 0$
2: Randomly initialize N population members to form S_t
3: **repeat**
4: Evaluate fitness of all members in S_t
5: **if** niching is desired **then**
6: calculate sharing function
7: **end if**
8: Perform N tournament selections on randomly chosen pairs from S_t to fill P_t
9: **if** mating restriction is desired **then**
10: calculate similarity measures of members in P_t
11: **end if**
12: Perform $N/2$ crossover operations on randomly chosen pairs from P_t with probability p_c to form P'_t
13: Perform mutation operation on each member of P'_t with probability p_m to form C_t
14: Evaluate fitness of all members in C_t
15: **if** elite preservation is desired **then**
16: Set S_{t+1} with the best N solutions of $P_t \cup C_t$
17: **else**
18: Set $S_{t+1} = C_t$
19: **end if**
20: $t \leftarrow t + 1$
21: **until** TP is satisfied
22: Declare population best as near-optimal solution

Another commonly used RGA is called the CHC (Cross-generational selection, Heterogeneous recombination and Cataclysmic mutation) [133] in which both parent and offspring population (of the same size N) are combined and the best N members are chosen. Such an algorithm can be realized by using an update plan which chooses the best N members from a combined $B \cup C$ population. The CHC algorithm also uses a mating restriction scheme.

Besides the classical and evolutionary algorithms, there exist a number of hybrid search and optimization methods in which each population member of a GA undergoes a local search operation (mostly using one of the classical optimization principles). In the so-called Lamarckian approach, the resulting solution vector replaces the starting GA population member, whereas in the Baldwinian approach the solution is unchanged but simply the modified objective function value is used in the subsequent search operations. Recent studies [134, 135] showed that instead of using a complete Baldwin or a complete Lamarckian approach to all population members, the use of Lamarckian approach to about 20 to 40% of the population members and the use of Baldwin approach to the remaining solutions is a better strategy. In any case, the use a local search strategy to update a solution can be considered as a special-purpose *mutation* operator associated with the generational plan of the proposed metaheuristic algorithm. Whether the mutated solution is accepted in the population (the Lamarckian approach) or simply the objective function value of the mutated solution is used (the Baldwin approach) or they are used partially (the partial Lamarckian approach) is a matter of implementation.

10. Genetic Programming (GP)

Genetic programming extends the concept of genetic algorithms from evolution of binary strings and real-valued variables to computer programs. It is defined as a 'weak' search algorithm for automatically generating computer programs to perform specified tasks. Weak search methods do not require specification of the form or structure of the solution in advance. A classic case study for such methods is the Santa Fe Trail problem [136]. An irregular twisting trail of food pellets is placed on a 32x32 grid. The trail consists of single and double gaps along straight regions and single, double and triple gaps at corners. An artificial ant starting at a given cell can move from cell to cell and consume these food pellets as it encounters them. The task is to generate a computer program for an ant which maximizes the food intake in a given number of steps (or minimize the number of steps required to consume all food pellets). Another common application of GP is seen in symbolic regression, where the regression function has no specified mathematical form and need to be derived from data.

GP systems evolve programs in a domain-specific language specified by primitives called *functions* and *terminals*. The terminal set ($\mathcal{T}$) may consist of the program's external inputs, random (ephemeral) constants and nullary (zero-argument) functions or operators or actions, whereas the function set ($\mathcal{F}$) may contain arithmetic operators ($+, -, \times$, etc.), mathematical functions ($\sin, \cos$, etc.), Boolean operators (`AND`, `OR`, etc.), conditional operators (`IF-THEN-ELSE`, etc.), programming constructs (`FOR`, `DO-WHILE`, etc.) or any other functions that are defined in the language being used. In symbolic regression problems, the function set contains mathematical and arithmetic operators and the terminal set consists of the independent variables and ephemeral constants. The Santa Fe Trail problem discussed above, has been successfully solved by Koza using actions `MOVE FORWARD`, `TURN RIGHT` and `TURN LEFT` as the terminals and `FOOD AHEAD?`, `PROGN2` and `PROGN3` as the functions. The `FOOD AHEAD?` function allows the ant to sense whether the cell directly in front of it contains food, based on which the program can make a decision. `PROGN2` and `PROGN3` are functions that take two and three sub-programs respectively and execute them in order.

The search space for such programs (or mathematical expressions) is immense and as such a metaheuristic is required to find a near-optimal solution. More importantly, the structure and length of the program are unknown, making it difficult to efficiently solve the problem with any common metaheuristic. The representation in GP however, allows evolution of programs. The programs are most often represented as trees, just as most compilers use parse trees internally to represent programs. Other common ways of expressing programs include linear [137] and graph-based representations, Parallel Distributed GP [138] and Cartesian GP [139].

A typical GP algorithm starts with a population of randomly created individuals or programs or trees. Two initialization methods are very common in GP, the `FULL` method and the `GROW` method [140]. The `FULL` method always generates trees in which all leaves (end nodes) are at the same user-specified depth value. This is achieved by randomly selecting nodes only from the $\mathcal{F}$ set until the depth limit is reached,

at which point nodes are selected from the $\mathcal{T}$ set. On the other hand, the `GROW` method creates trees of varied sizes and shapes by randomly selecting nodes from the full primitive set ($\mathcal{F} \cup \mathcal{T}$) for all nodes until the depth limit is reached, at which point nodes are chosen from $\mathcal{T}$ as in the case of `FULL` method. Next the fitness for each individual is determined by 'running' the program. For example, in case of Santa Fe Trail problem, GP executes the trail that each individual represents and calculates the number of food pellets consumed by the ant. High fitness individuals are selected to form the mating pool, on which primary genetic operations, namely crossover and mutation, are applied to create a new population of programs. In GP, sub-tree crossover refers to exchange of randomly selected (but compliant) parts of the parent programs. Mutation usually involves randomly replacing one or more functions or terminals with other compliant primitives. The process is repeated until some stopping criterion (like maximum number of generations) is met.

Algorithm 11 shows the pseudocode for a generic GP implementation. Note its structural similarity to Algorithm 10. Niching, mating restriction and elite preservation operations are also relevant to GP. Additionally, GP also uses automatically defined functions (ADFs) [141] and architecture-altering operations [142], both introduced by Koza. ADFs are sub-programs that are dynamically evolved during GP and can be called by the program being evolved. ADFs contains a set of reusable program statements which occur together frequently in the main program. For example, an ADF for calculating the scalar product of two vectors can greatly simplify the evolution of a program for calculating the product of two matrices. Automatically defined iterations (ADIs), automatically defined loops (ADLs) and automatically defined recursions (ADRs) have also been proposed in [142]. ADFs can either be attached to specific individuals in the population, as suggested by Koza, or be provided as a dynamic library to the population [143, 144].

Architectural-altering operators, like argument and sub-routine creation/deletion, act upon the program and its automatically defined components to modify their architecture, something that is not possible through crossover or mutation. Modern GP implementations also use bloat control mechanisms. Bloat refers to a non-functional or redundant part of the program being evolved which grows with generations [145]. Bloating is undesirable because it increases GP's computational time. The most common way of reducing bloat is to constrain program size (number of tree nodes n and its depth d). Other methods use some measure of parsimony or program complexity [146]. A comparison of these methods can be found in [147].

GP has been applied in several areas other than automatic programming, such as automatic synthesis, machine learning of functions, scientific discovery, pattern recognition, symbolic regression (also known as model induction or system identification) and art. Readers are referred to [148] for a thorough review.

11. Particle Swarm Optimization (PSO)

The canonical form of PSO consists of a population of particles, known as a swarm, with each member of the swarm being associated with a position vector $\mathbf{x}_t$ and a velocity vector $\mathbf{v}_t$. The size of these vectors is equal to the dimension of the search space. The term velocity ($\mathbf{v}_t$) at any iteration t indicates the directional distance that the particle has covered in the $(t-1)$-th iteration. Hence, this term can also be called as a *history* term. The velocity of the population members moving through the search space is calculated by assigning stochastic weights to $\mathbf{v}_t$ and the attractions from a particle's personal-best or 'pbest' ($\mathbf{p}_l$) and swarm's best or 'gbest' ($\mathbf{p}_g$), and computing their resultant vector. The 'pbest' indicates the best position attained by a particle so far, whereas 'gbest' indicates the best location found so far in the entire swarm (this study implements popularly used fully informed swarm topology, i.e. each particle knows the *best* location in the entire swarm rather than in any defined neighborhood).

The following equations describe the velocity and position update for i-th particle at any iteration t (we refer to this as the *child creation rule*):

$$\mathbf{v}_{i,t+1} = w\mathbf{v}_{i,t} + c_1\mathbf{r}_1 \mathbin{.*} (\mathbf{p}_{l,i} - \mathbf{x}_{i,t}) + c_2\mathbf{r}_2 \mathbin{.*} (\mathbf{p}_g - \mathbf{x}_{i,t}), \tag{30}$$

$$\mathbf{x}_{i,t+1} = \mathbf{x}_{i,t} + \mathbf{v}_{i,t+1}. \tag{31}$$

Algorithm 11 Genetic Programming

Require: N, p_c, p_m, n, d

Set iteration counter $t = 0$
Randomly initialize N population members (trees) to form S_t
if ADFs are used **then**
  Attach randomly initialized ADFs to the population members
end if
repeat
  Evaluate fitness of all members in S_t
  if niching is desired **then**
    calculate sharing function
  end if
  Perform N tournament selections on randomly chosen pairs from S_t to fill P_t
  if mating restriction is desired **then**
    calculate similarity measures of members in P_t
  end if
  Set $C_t = \Phi$
  for $i = 1$ **to** $N/2$ **do**
    Choose two random individuals $I_{1,t}$ and $I_{2,t}$ from P_t
    if $rand(0,1) \leq p_c$ **then**
      if ADFs are used **then**
        Perform sub-tree crossover on ADF parts of $I_{1,t}$ and $I_{2,t}$ to yield offspring $I'_{1,ADF}$ and $I'_{2,ADF}$
        Perform sub-tree crossover on non-ADF parts of $I_{1,t}$ and $I_{2,t}$ to yield offspring $I'_{1,NADF}$ and $I'_{2,NADF}$
        Combine $I'_{1,ADF}$ and $I'_{1,NADF}$ to yield $I'_{1,t}$
        Combine $I'_{2,ADF}$ and $I'_{2,NADF}$ to yield $I'_{2,t}$
      else
        Perform sub-tree crossover on $I_{1,t}$ and $I_{2,t}$ to yield $I'_{1,t}$ and $I'_{2,t}$
      end if
    else
      Set $I'_{1,t} = I_{1,t}$ and $I'_{2,t} = I_{2,t}$
    end if
    if $rand(0,1) \leq p_m$ **then**
      Perform mutation on $I'_{1,t}$ to yield $I_{1,t+1}$
      Perform mutation on $I'_{1,t}$ to yield $I_{1,t+1}$
    else
      Set $I_{1,t+1} = I'_{1,t}$ and $I_{2,t+1} = I'_{2,t}$
    end if
    if architectural-altering operations are used **then**
      Apply randomly chosen operations to ADF parts of $I_{1,t+1}$ and $I_{2,t+1}$
    end if
    $C_t = C_t \cup \{I_{1,t+1}, I_{2,t+1}\}$
  end for
  if bloat control is desired **then**
    Apply size constraints $\{n, d\}$ or parsimony pressure based measures
  end if
  Evaluate fitness of all members in C_t
  if elite preservation is desired **then**
    Set S_{t+1} with the best N solutions of $P_t \cup C_t$
  else
    Set $S_{t+1} = C_t$
  end if
  $t \leftarrow t + 1$
until TP is satisfied
Declare population best as near-optimal solution

Here, $\mathbf{r}_1$ and $\mathbf{r}_2$ are random vectors (with each component in $[0, 1]$), and w, c_1 and c_2 are pre-specified constants. The $.*$ signifies component-wise multiplication of two vectors. In each iteration, every particle in the population is updated serially according to the above position and velocity rules.

A little thought will reveal that the above child creation rule involves a population member $\mathbf{x}_i$ at the current generation, its position vector in the previous generation, and its best position so far. We argue that these vectors are all *individualistic* entities of a population member. The only non-individualistic entity used in the above child creation is the position of the globally best solution $\mathbf{p}_g$, which also justifies the population aspect of the PSO. The standard PSO is shown in Algorithm 12.

Algorithm 12 Particle Swarm Optimization Algorithm

Require: N, w, c_1, c_2
1: Set iteration counter $t = 0$
2: Randomly initialize positions $\mathbf{x}_t$ and velocities $\mathbf{v}_t$ of N particles to form S_t
3: **for** $i = 1$ **to** N **do**
4: Set $\mathbf{p}_{l,i} = \mathbf{x}_{i,t}$
5: **end for**
6: Evaluate fitness $f(\mathbf{x}_t)$ of all particles in S_t
7: Assign best fitness particle as $\mathbf{p}_g$
8: **repeat**
9: **for** $i = 1$ **to** N **do**
10: Calculate new velocity for particle i using Equation (30) or Equation (32)
11: Calculate new position for particle i using Equation (31)
12: **if** $f(\mathbf{x}_{i,t+1}) < f(\mathbf{p}_{l,i})$ **then**
13: Set $\mathbf{p}_{l,i} = \mathbf{x}_{i,t+1}$
14: **end if**
15: **if** $f(\mathbf{x}_{i,t+1}) < f(\mathbf{p}_g)$ **then**
16: Set $\mathbf{p}_g = \mathbf{x}_{i,t+1}$
17: **end if**
18: Update particle i's position and velocity to $\mathbf{x}_{i,t+1}$ and $\mathbf{v}_{i,t+1}$
19: **end for**
20: $t \leftarrow t + 1$
21: **until** TP is satisfied
22: Declare swarm best as near-optimal solution

The parametric study on coefficients (w, c_1, and c_2) for terms $\mathbf{v}_{i,t}$, $(\mathbf{p}_{l,i} - \mathbf{x}_{i,t})$ and $(\mathbf{p}_g - \mathbf{x}_{i,t})$, respectively, were conducted in [149, 150] and empirical studies revealed that in Equation (31), the w value should be about 0.7 to 0.8, and c_1 and c_2 around 1.5 to 1.7. Irrespective of the choice of w, c_1 and c_2, while working with the bounded spaces the velocity expression often causes particles to 'fly-out' of the search space. To control this problem, a velocity clamping mechanism was suggested in [151]. The clamping mechanism restricted the velocity component in $[-v_{\max,j}\ v_{\max,j}]$ along each dimension (j). Usually, $v_{\max,j}$ along j-th dimension is taken as 0.1 to 1.0 times the maximum value of x_i along the i-th dimension. Such a clamping mechanism does not necessarily ensure that particles shall remain in the search space.

The velocity term ($\mathbf{v}_{i,t}$) indicates a particle's ability to explore the search space while it moves under the attraction of 'pbest' and 'gbest'. In initial phases of the search, wide explorations are favored, whereas towards the end more focused search is desired. To achieve this, the concept of decreasing inertia weight (w) was introduced in [152]. The strategy has gained popularity in promoting convergence. The idea of varying coefficients was also extended successfully to dynamically update the parameters c_1 and c_2 in [153, 154].

Premature convergence has been a major issue in PSO. The particles often accelerate towards the swarm's best location and the population collapses. A study classified under swarm stability and explosion

[155] proposed a velocity update rule based on 'constriction factor' (χ). In the presence of χ, the velocity update Equation (30) becomes:

$$\mathbf{v}_{i,t+1} = \chi \left(\mathbf{v}_{i,t} + c_1 \mathbf{r}_1 \ .* \ (\mathbf{p}_{l,i} - \mathbf{x}_{i,t}) + c_2 \mathbf{r}_2 \ .* \ (\mathbf{p}_g - \mathbf{x}_{i,t}) \right). \tag{32}$$

This is one of the most popular versions of velocity update rule in PSO studies.

12. Scatter Search (SS)

Scatter search was first introduced by Fred Glover in 1977 [156] to solve integer programming problems. Later the basic SS method was enhanced to solve other optimization problems. The original SS method was developed using five different operators, as described below:

Diversification operator: In this operation, a set of diverse solutions is generated using a seed solution.

Improvement operator: In this operation, a solution is transformed into one or more enhanced solutions. This is similar to a local search heuristic. If the operator is unable to produce a better solution, the original solution is considered as the enhanced solution.

Reference set update operator: A set of b best solutions are collected through this operator. This set can be considered as an 'elite set'. Solutions gain membership to the reference set based on their quality and/or their diversity within the reference set.

Subset generation operator: This operator takes the reference set and chooses a subset for performing the solution combination operator described next.

Solution combination operator: This operator transforms a given subset of solutions produced by the subset generation operator into one or more combined solutions.

The SS methodology works as follows. Initially, a set S_0 of N solutions is created using a combination of diversification operator and the improvement operator. Then, reference set update operator is used to select the best b solutions in terms of objective values and their diversity from each other and saved into a reference set RS. Usually, $|RS| = N/10$. One way to use both objective function value and diversity measure is to select b_1 solutions (set B_1) from the top of the list ranked according to objective values (of S_0) and then select b_2 solutions (set B_2) using the maximum of minimum distances of unselected solutions in S_0 from the chosen reference set B_1. In this case, $b = b_1 + b_2$, where b_1 and b_2 can be chosen appropriately to emphasize convergence and diversity maintenance, respectively, of the resulting algorithm. They can also be made adaptive by monitoring different performance measures during iterations.

New subsets of solutions are then created using the subset generation operator. In its simplest form, every pair of solutions from RS is grouped as a subset, thereby making a total of $\binom{b}{2}$ or $b(b-1)/2$ subsets. Then, the solution combination operator is applied to each subset one at a time to obtain one or more combined solutions. Each of these new solutions are then modified using the improvement operator. Again the reference set update operator is used to choose b best solutions from combined set of RS and improved solutions. If the new reference set is identical to previous reference set after all subsets are considered, this means that no improvement in the reference set is obtained and the algorithm is terminated. On the other hand, if at least one subset generates a new solution that makes the reference set better than before, the algorithm is continued. The subset combination operator can use a recombination-like operator (which may introduce user parameters) by using two solution vectors to create a single or multiple solutions. The above generic SS procedure is shown in Algorithm 13.

More sophisticated methodologies for the above five operators exist, including path relinking [157], dynamic reference set updating, reference set rebuilding etc. [158]. The primary reference for the SS approach is the book by Laguna and Marti [159].

Algorithm 13 Scatter Search

Require: N, b_1, b_2

Set iteration counter $t = 0$
Set $RS = \Phi$
Apply diversification operator to generate N solutions to form S_t
Apply improvement operator on all solutions in S_t
repeat
  Evaluate fitness of all members in S_t
  Set $B_1 = \Phi$, $B_2 = \Phi$
  Find b_1 best fitness solutions in $RS \cup S_t$ to form B_1
  Evaluate minimum distance d for each solution in $(RS \cup S_t) - B_1$ from set B_1
  Find b_2 solutions with highest d values to form B_2
  Set $RS = B_1 \cup B_2$
  Apply subset generation operator on RS to form new solution set C'_t
  Apply improvement operator on all solutions in C'_t to form C_t
  Set $S_t = C_t$
  $t \leftarrow t + 1$
until TP is satisfied
Declare best solution in RS as near-optimal solution

13. Simulated Annealing (SA)

The simulated annealing (SA) method resembles the cooling process of molten metals through a structured annealing process. At a high temperature, the atoms in the molten metal can move freely with respect to each another, but as the temperature is reduced, the movement of the atoms gets restricted. The atoms start to get ordered and finally form crystals having the minimum possible energy. However, the formation of the crystal mostly depends on the cooling process and cooling rate. If the temperature is reduced at a very fast rate, the crystalline state may not be achieved at all, instead, the system may end up in a polycrystalline state, which may have a higher energy state than that in the crystalline state. Therefore, in order to achieve the absolute minimum energy state, the temperature needs to be reduced at a slow rate. The process of slow cooling is known as *annealing* in metallurgical parlance.

The simulated annealing procedure simulates this process of slow cooling of molten metal to achieve the minimum function value in a minimization problem. The cooling phenomenon is simulated by controlling a temperature-like parameter introduced with the concept of the Boltzmann probability distribution. According to the Boltzmann probability distribution, a system in thermal equilibrium at a temperature T has its energy distributed probabilistically according to $P(E) = \exp(-E/kT)$, where k is the Boltzmann constant. This expression suggests that a system at a high temperature has almost uniform probability of being at any energy state, but at a low temperature it has a small probability of being at a high energy state. Therefore, by controlling the temperature T and assuming that the search process follows the Boltzmann probability distribution, the convergence of an algorithm can be controlled. Metropolis, et al. [160] suggested one way to implement the Boltzmann probability distribution in simulated thermodynamic systems. The same can also be used in the function minimization context. Let us say, at any instant the current point is $s^{(t)}$ and the function value at that point is $E(t) = f(s^{(t)})$. Using the Metropolis algorithm, we can say that the probability of the next point being at $s^{(t+1)}$ depends on the difference in the function values at these two points or on $\Delta E = f(s^{(t+1)}) - f(s^{(t)})$ and is calculated using the Boltzmann probability distribution:

$$P(E(t+1)) = \min\left[1, \exp\left(-\Delta E/kT\right)\right].$$

If $\Delta E \leq 0$, this probability is one and the point $s^{(t+1)}$ is always accepted. In the function minimization context, this makes sense because if the function value at $s^{(t+1)}$ is better than that at $s^{(t)}$, the point $s^{(t+1)}$

must be accepted. The interesting situation happens when $\Delta E > 0$, which implies that the function value at $s^{(t+1)}$ is worse than that at $s^{(t)}$. According to many traditional algorithms, the point $s^{(t+1)}$ must not be chosen in this situation. But according to the Metropolis algorithm, there is some finite probability of selecting the point $s^{(t+1)}$ even though it is a worse than the point $s^{(t)}$. However, this probability is not the same in all situations. This probability depends on relative magnitude of ΔE and T values. If the parameter T is large, this probability is more or less high for points with largely disparate function values. Thus, any point is almost acceptable for a large value of T. On the other hand, if the parameter T is small, the probability of accepting an arbitrary point is small. Thus, for small values of T, the points with only small deviation in function value are accepted.

Simulated annealing is a point-by-point method. The algorithm begins with an initial point and a high temperature T. A second point is created at random in the vicinity of the initial point and the difference in the function values (ΔE) at these two points is calculated. If the second point has a smaller function value, the point is accepted; otherwise the point is accepted with a probability $\exp(-\Delta E/T)$. This completes one iteration of the simulated annealing procedure. In the next generation, another point is created at random in the neighborhood of the current point and the Metropolis algorithm is used to accept or reject the point. In order to simulate the thermal equilibrium at every temperature, a number of points (n) is usually tested at a particular temperature, before reducing the temperature. The algorithm is terminated when a sufficiently small temperature is obtained or a small enough change in function values is found. Algorithm 14 shows the pseudocode for simulated annealing.

Algorithm 14 Simulated Annealing

Require: Cooling schedule $g(T)$, n
1: Set iteration counter $t = 0$
2: Initialize random solution s_0
3: Initialize temperature T to a very high value
4: Set $s_{best} = s_0$
5: **repeat**
6: Evaluate fitness of s_t
7: **repeat**
8: Create s_{t+1} randomly in the neighborhood of s_t
9: Set $\Delta E = f(s_{t+1}) - f(s_t)$
10: **until** $\Delta E < 0$ **or** $rand(0,1) \leq \exp(-\Delta E/T)$
11: **if** $(t \mod n) = 0$ **then**
12: Update $T = g(T)$
13: **end if**
14: **if** $f(s_{t+1}) < f(s_t)$ **then**
15: $s_{best} = s_{t+1}$
16: **end if**
17: $t \leftarrow t + 1$
18: **until** TP is satisfied
19: Declare s_{best} as near-optimal solution

The initial temperature (T) and the number of iterations (n) performed at a particular temperature are two important parameters which govern the successful working of the simulated annealing procedure. If a large initial T is chosen, it takes a number of iterations for convergence. On the other hand, if a small initial T is chosen, the search is not adequate to thoroughly investigate the search space before converging to the true optimum. A large value of n is recommended in order to achieve quasi-equilibrium state at each temperature, but the computation time is more. Unfortunately, there are no unique values of the initial temperature and n that work for every problem. However, an estimate of the initial temperature can be obtained by calculating the average of the function values at a number of random points in

the search space. A suitable value of n can be chosen (usually between 20 to 100) depending on the available computing resource and the solution time. Nevertheless, the choice of the initial temperature and subsequent cooling schedule still remain an art and usually require some trial-and-error efforts.

14. Tabu Search (TS)

Tabu search algorithm proposed by Glover [161] is a popular single-solution metaheuristic. It is a sophisticated version of the simulated annealing algorithm, in which a memory of past solutions is kept and used to prevent the algorithm from revisiting those solutions (cycling) when a 'non-improving' move is made to escape local optima. For computational tractability, this memory is basically a 'tabu list' of past moves (instead of the solutions themselves) which should be prevented in the following iterations. Each move in the tabu list is forbidden for a given number of iterations, called the tabu tenure (τ). Tabu solutions may be accepted if they meet certain conditions known as *aspiration criteria*. A commonly used criterion it to go ahead with a forbidden move if it generates a solution better than the best found solution.

A TS algorithm starts with a single solution like in simulated annealing and N new solutions are sampled from its neighborhood. Solutions that require a forbidden move are replaced by other samples. The best of these neighborhood solutions is chosen and if its fitness is better than the current solution, then the former replaces the latter, just like in any local search algorithm. However, if none of the neighborhood solutions are better than the current solution, a non-improving move is made. In simulated annealing this move is a controlled randomization. In TS, a move that is not forbidden by the tabu list is performed instead to prevent cycles. The size of the tabu list k puts an upper bound on the size of the cycle that can be prevented. The size of k will determine the computational complexity of the algorithm. Sometimes multiple tabu lists may also be used to capture other attributes of visited solutions like variables that change values (continuous search space) or elements that change positions (discrete search space) [162].

In addition to the tabu list, which is categorized as short-term memory, a TS can also have a medium-term memory which controls search intensification and a long-term memory which enables search diversification. The purpose of the medium-term memory is to utilize the information of the recorded elite solutions to guide the search process towards promising regions of the search space. One way to achieve this is to find features that are common to recorded elite solutions and force the common features to exist in the newly created solution. This process is called the intensification. The purpose of the long-term memory is to introduce diversity in the search space by creating solutions in the relatively unexplored regions of the search space. The frequency of occurrence of different regions of the search space is recorded in a memory and a new solution is created in the region having the least frequency of occurrence thus far. This is called diversification.

TS has been found to be efficient in solving combinatorial optimization problems having discrete search spaces [163, 164] and vehicle routing [165]. The bare framework of TS is shown in Algorithm 15. Actual implementations of the three types of memory can be found in [166].

15. Other Metaheuristic Techniques

A host of other metaheuristic optimization methods inspired by nature, biological and physical phenomena have been suggested in the recent past. In this section we have attempted to give an exhaustive list of these metaheuristics without going into implementation specifics and instead very briefly describe their basis. We also specify the number of citations as of June 2015 of the oldest paper that we could find on Google Scholar which describes the methodology. This number can be thought of as an indicator of the popularity and impact of the method in the field of metaheuristic search and optimization.

1. **Harmony Search [167, 168]** (2119): The analogy with jazz musicians adjusting pitches to attain harmony (musical aesthetics) is used to generate offspring, first by discrete recombination and then through mutation.

Algorithm 15 Tabu Search

Require: N, τ, k

Set iteration counter $t = 0$
Initialize random solution s_0
Initialize tabu list T, medium-term memory M and long-term memory L
Set $s_{best} = s_0$
repeat
  Evaluate fitness of s_t
  Set $\mathcal{N}(s_t) = \Phi$
  repeat
    Create a neighbor to s_t
    if move to neighbor occurs in T **and** aspiration criteria is **false then**
      Discard the neighbor
    else
      Add neighbor to $\mathcal{N}(s_t)$
    end if
  until $|\mathcal{N}(s_t)| = N$ **or** no more neighbors exist
  Find best solution s'_t in $\mathcal{N}(s_t)$
  if $f(s'_t) < f(s_t)$ **then**
    $s_{best} = s'_t$
    Delete moves older than τ iterations from T
    Add move $s_t \Rightarrow s'_t$ to T
    while $size(T) > k$ **do**
      Delete oldest move in T
    end while
  end if
  $s_t = s'_t$
  if intensification is desired **then**
    Modify s_t
    Update M
  end if
  if diversification is desired **then**
    Modify s_t
    Update L
  end if
  $t \leftarrow t + 1$
until TP is satisfied
Declare s_{best} as near-optimal solution

2. **Estimation of Distribution Algorithms [169, 170]** (1912): This method uses a probabilistic model of the current population to generate offspring instead of using crossover or mutation operators. The best of the offspring population are selected to update the probability distribution, which is usually either a Bayesian network or a multivariate normal distribution.
3. **Bacterial Foraging [171]** (1662): The biology and physics underlying the chemotactic (foraging) behavior of *E. coli* bacteria are modeled through operators in optimization that mimic chemotaxis (guided movement), swarming (co-operation), reproduction, elimination (death) and dispersal (random movement).
4. **Memetic Algorithms [172]** (1445): A hybridization of population-based metaheuristics (usually GA), local search heuristics, problem specific information and sometimes machine learning is used to evolve a population of solutions.
5. **Gene Expression Programming [173]** (1435): It is a variant of linear genetic programming where fixed length genes are encoded using left to right and top to bottom traversal of parse trees.
6. **Coevolutionary Algorithms [174, 175]** (1204): The analogy with species (sub-populations) in an ecosystem is used so that each sub-population represents a component of the solution, evolved independently by a standard GA.
7. **Firefly Algorithm [176, 177]** (1182): The analogy with fireflies using light to communicate with their neighbors is used, where each firefly (solution) moves towards brighter fireflies (brightness proportional to the fitness function) based on the distance between them and the brightest firefly moves randomly.
8. **Gravitational Search Algorithm [178]** (1052): It uses a gravitationally isolated system of objects (solutions) in the search space with masses proportional to the fitness function and with accelerations and velocities determined by formulae loosely based on the law of gravitation.
9. **Cuckoo Search [179]** (985): The analogy with *brood parasitism* and *Lévy flight* behavior seen in cuckoo birds is used. Each cuckoo performs a Lévy flight to reach a new location and lays an egg(s) (solution) which replaces a randomly chosen egg in a randomly chosen nest depending on its fitness. Host birds can either throw away (discard) an egg or abandon their nests to build new ones.
10. **Biogeography-based optimization [180]** (907): The analogy with migration of species is used to assign emigration and immigration probabilities for each habitat (solution) based on its Habitat Suitability Index (fitness), and these probabilities decide the movement of species between habitats thus modifying them.
11. **Shuffled Frog Leaping Algorithm [181, 182]** (875): It is a variant of memetic algorithm, where a population of frogs (solutions) is organized into different groups (memeplexes) within which a PSO type search is performed locally, following which the memeplexes are shuffled.
12. **Bees Algorithm [183]** (787): It is similar to ABCO with the difference that the former uses elitist strategy to assign more neighborhood-searching bees to good fitness sites, following which the best bee from each path is selected to form new bees.
13. **Cultural Algorithms [184]** (747): The analogy with human social evolution is used to evolve a population using a *belief space* which itself gets adjusted by high-performing solutions in the population.
14. **Imperialist competitive algorithm [185, 186]** (727): The analogy with imperialism and colonization is used, where powerful imperialist countries (high-fitness solutions) are assigned colonies (other solutions) which can 'move' towards their center of power and can even exchange places with it if a better location is found. Powerful empires (imperialists and their colonies) try to capture the weaker ones by competing with each other (proportionate probabilistic selection) in terms of their combined power (weighted fitness).
15. **Bat Algorithm [187]** (532): It mimics the echolocation ability of microbats. Each microbat (solution) is associated with a velocity and a frequency of emitted sound, which it uses to move through the search space. When a prey (better fitness location) is located, the sound emitted by the bat are modified by decreasing the amplitude and increasing the pulse emission rate.

16. **Grammatical Evolution [188, 189]** (463): It uses the Backus-Naur form of context-free grammar to encode programs as integer (and hence binary) strings which can be evolved by most metaheuristics, thus making automatic programming, language independent.
17. **Invasive Weed Optimization [190]** (425): It mimics robustness, adaptation and randomness of colonizing weeds. A fixed number of seeds (solutions) are dispersed in the search space. Each seed grows into a flowering plant and disperses its own seeds in the neighborhood based on its fitness. After a maximum number of plants is reached, weaker ones are eliminated in favor of stronger ones.
18. **Charged System Search [191]** (291): Solutions are represented by charged particles associated with positions and velocities, the amount of charge representing the fitness value. A highly charged particle can attract any other particle with a lower charge. However, only some low charge particles can attract particles with higher charge. The Coulomb's law is used to calculate the forces of attraction between particles and their positions and velocities are adjusted.
19. **Big Bang-Big Crunch Optimization [192]** (290): This algorithm is designed to simulate big bang and big crunch occurring one after the other in each iteration. The first big bang generates random solutions. The following big crunch converges all solutions to the center of mass where mass is proportional to the fitness value. Thereafter, each big bang generates new solutions around this center of mass using a normal distribution. The solutions eventually collapse to the optimum.
20. **Marriage in Honey Bees Optimization [193]** (271): It models the mating flight of queen bees as seen in bee colonies. The queen bee moves based on its speed, energy and position and collects genetic material (solution coordinates) from good fitness drones. Upon reaching the colony it generates broods by combining its genome with a randomly selected drone genome (crossover). The best broods replace the worst queens and the remaining broods are killed.
21. **Bee Colony Optimization [194]** (261): It uses a similar analogy to ABCO described in Section 4. In the *forward pass*, employer bees fly out of the colony and perform a series of moves to construct partial solutions. In the *backward pass*, they return to the colony and share information with other bees. Based on the collective information from all bees, each bee can either abandon its partial solution, continue to construct its partial solution or recruit bees to help further explore its partial solution.
22. **Group Search Optimizer [195]** (257): It uses the analogy with the producer-scrounger model of animal foraging. The producer (best fitness member) performs *scanning* to find a better resource location. The scanning strategy mimics the *white crappie* fish, which uses a series of cone shaped visual scans to locate food. Other members (scroungers) perform random walk towards the producer and can become a producer if they find a better fitness location along the way.
23. **Honey-bee Mating Optimization [196]** (204): Like marriage in honey bees optimization, this metaheuristic also mimics the mating flight of queen bees of a honey bee colony. However, it can also handle continuous variables.
24. **Krill Herd Algorithm [197]** (196): It is based on the herding behavior of krill individuals. The movement of each krill individual depends on the minimum of its distance from food (foraging) and from highest density of the herd (herding). Additionally, random movement is also used to enhance exploration.
25. **BeeHive Algorithm [198]** (182): The metaheuristic was developed to solve routing problems using *short distance bee agents* for neighborhood search and *long distance bee agents* for global exploration.
26. **Glowworm Swarm Optimization [199]** (170): Similar to ACO, this metaheuristic uses glowworms instead of ants as agents. The glowworms emit light whose intensity is proportional to the amount of *luciferin* (fitness value) they have. Brighter glowworms act as attractor to other glowworms given that the former are within the sensor range of the latter. This approach allows multimodal function optimization.
27. **Intelligent Water Droplet Algorithm [200]** (163): It is a swarm-based metaheuristic inspired from water drops carrying soil particles in rivers. In combinatorial problems, each water drop

gradually builds a solution as it moves and increases its velocity according to the soil content of the path. More soil corresponds to a lower increase in velocity. Over iterations, paths with lesser soil content are preferred probabilistically.

28. **Bee System [201]** (133): It adopts the honey bee colony analogy to solve combinatorial problems. For example, in the traveling salesman problem, the nectar quantity represents the length of the path. The bees are allowed to fly to any node with a probability that decreases with increasing distance from their current location. The influence of this distance increases with iterations, making search more random at the beginning and more restricted towards the end.
29. **Stochastic Diffusion Search [202]** (120): It uses agents with hypotheses communicating directly with each other. In the *test phase*, each agent evaluates its hypothesis and finds it to be either successful or unsuccessful. In the *diffusion phase*, agents with unsuccessful hypotheses either select other agents to follow or generate new hypotheses.
30. **Cat Swarm optimization [203]** (108): As the name suggests, the algorithm attempts to model the behavior of cats which represent solutions having position and velocity. Each cat has two modes of movement in the search space: the seeking mode and the tracking. The former plays the role of the local search component of the metaheuristic while the later tends to global search.
31. **Central Force Optimization [204]** (96): It also uses the analogy with particle motion under gravitational force like gravitational search algorithm, although with different formulations. Central force optimization is inherently deterministic unlike most other metaheuristics.
32. **Queen-bee Evolution [205]** (95): It is basically an integration of queen bee's mating flight behavior with genetic algorithms. Instead of randomly selecting parents for crossover, each individual mates with the queen bee (best fitness individual of previous generation). The mutation operation is also modified.
33. **Cooperative Bees Swarm Optimization [206]** (94): Unlike other bee algorithms, this method starts with a single bee which provides the best location based on neighborhood search. Other bees are then recruited to a specified region around the best known location. After performing neighborhood searches, all bees communicate their individual best locations to the swarm. A tabu list of visited locations is also maintained to avoid cycles.
34. **Artificial Fish Swarm Optimization [207]** (91): It tries to mimic the social behavior of fishes such as preying for food, swarming and following their neighbors. Each fish performs a local search by *visually* examining its surroundings and moving towards a better food source (objective value). It may also either swarm another fish's food source or simply follow one of its neighbors.
35. **Eagle Strategy [208]** (91): It uses the foraging behavior of eagles as a metaheuristic. Eagles representing candidate solutions initially perform a slow search where they search a larger area using differential evolution. Once promising regions (prey) are identified the chasing phase begins which uses a fast and local search algorithms such as hill-climbing methods.
36. **Differential Search Algorithm [209]** (89): This algorithm simulates the Brownian-like random-walk movement used by various organisms to migrate as a superorganism. Each superorganism represents a candidate solution and each of its members corresponds to one variable dimension. As superorganisms move in the search space they exchange information about their best location with other superorganisms.
37. **Backtracking Search Optimization [210]** (74): It is a simplified form of evolutionary algorithm which uses the following operations: initialization, selection-1, mutation, crossover and selection-2. Selection-1 uses the best of the historical population members to generate a population. Mutation mutates the current population using the output of selection-1. Crossover recombines the best individuals of the mutant population and finally selection-2 uses greedy selection to update the current population with better offspring if found.
38. **Flower Pollination Algorithm [211]** (61): Each solution is represented by a pollen gamete from a flowering plant. Flower pollen are carried by pollinators such as insects which move according to Lévy flights. All gametes either combine with the fittest pollen (global pollination) with a given probability or randomly combine with other pollen gametes (local pollination).

39. **Analytic Programming [212]** (58): This metaheuristic is capable of generating programs and can be used for symbolic regression. Unlike genetic programming, it can be used with any evolutionary algorithm since it acts as a superstructure requiring no changes to the variation operators.
40. **Fish School Search [213, 214]** (54): It uses the analogy with feeding, swimming and breeding behavior found in fish schools. Feeding of a fish is analogous to a candidate solutions' fitness value. Thus, the fish can gain or lose weight depending on the regions it swims in. Swimming and breeding refer to the actual search process where solutions swarm potential regions.
41. **Black Hole Optimization [215]** (54): It mimics the gravitation interaction of a black hole with nearby stars. The best solution in the current population becomes the black hole and attracts other solutions. Other stars may become a black hole if they reach a point with better fitness than the current black hole. When a solution passes the event horizon of the current black hole, it vanishes and a new star (solution) is randomly initialized.
42. **Monkey Search [216]** (51): The solutions are analogous to monkeys and undergo climb, watch-jump and somersault steps. The climb step acts like a local optimizer for each monkey. The watch-jump step represents exchange of fitness landscape information and the somersault step enable exploration of new regions in the search space.
43. **Bacterial Swarming Algorithm [217]** (50): This metaheuristic uses the tumble-and-run behavior of a bacterium's chemotactic process. The tumble operation acts like the global search component, where each bacterium (solution) moves independently to find good regions. Thereafter, the run operation causes local search.
44. **Water Cycle Algorithm [218]** (49): It is loosely based on the hydrologic cycle which consists of precipitation (random initialization of solutions), percolation into the sea (attraction towards the best solution), evaporation and again precipitation (reinitialization of some solutions).
45. **Gaussian Adaptation [219]** (49): Similar in principle to CMA-ES, this method samples single candidate solutions from a multivariate normal distribution and iteratively updates its mean and covariance so as to maximize the entropy of the search distribution. Strategy parameters are either fixed or adapted as in CMA-ES.
46. **River Formation Dynamics [220]** (45): It is used to solve the traveling salesman problem using the analogy with a flowing river eroding soil from high altitudes and depositing it at lower altitudes. Initially all nodes are at the same altitude. But as the water drops move from origin to destination they erode various edges until all drops follow the same path.
47. **Brain Storm Optimization [221]** (40): The solutions are considered as individuals coming up with different ideas during a brain storming session. The best ideas are selected and new ideas are created based on their combinations. Individuals with rejected ideas try to come up with new ideas (reinitialization).
48. **Roach Infestation Optimization [222]** (39): It mimics the collective behavior of cockroaches in finding dark regions in an enclosed space. The level of darkness is proportional to the fitness value. Initially each roach (solution) acts individually to find a dark region. However, when it comes in close proximity to another roach they exchange information about their respective regions. The information of the darkest known location will thus spread through the roach population.
49. **League Championship Algorithm [223]** (35): It is based on the metaphor of sporting competitions in sport leagues. Each candidate solution is a team and its position in the search space defines the team's strategy. Each week (iteration) the teams participate in pairwise competitions and based on their ranking modify their strategies.
50. **Electro-magnetism Optimization [224]** (31): It is similar in principle to charged system search. Charge particles represent solutions which move under the influence of electrostatic forces. The nature of the force on a particle (attractive or repulsive) due to another particle depends on the charge (fitness) of the other particle. Therefore, the motion does not strictly conform to physical electrostatic laws.

51. **Galaxy-based Search [225]** (25): It uses the metaphor of the spiraling arms of a galaxy sweeping the solution space to find better regions. This step is referred to as the spiral chaotic move. Thereafter, local search is performed within these regions to fine-tune the obtained solutions.
52. **Dolphin Echolocation [226]** (24): It mimics the echolocation ability of dolphins to find prey. Solutions (echoes) are initially randomly distributed in the search space. The fitness (echo quality) at each location is evaluated and more or less echoes are allocated in the neighborhood of each location based on the fitness values.
53. **Wolf Search Algorithm [227]** (20): The analogy with wolfs hunting for prey in packs is used in this algorithms. Each wolf (solution) has a fixed visual area and can sense companions within it. The quality of a wolf's position is represented by the fitness value of the solutions. A wolf which has a better placed companion in its sight moves towards it. At times, a wolf may sense an enemy and sprint to a random location (reinitialization).
54. **Artificial Cooperative Search [228]** (20): It uses two populations referred to as superorganisms. Each superorganism is a collection of sub-superorganisms (solutions) and each sub-superorganism is a collection of variable dimensions (i.e. a solution). The sub-superorganisms have a predator-prey relationship based on the relative fitness values. Whether a superorganism is the prey or the predator is determined randomly in each generation.
55. **Rubber Band Technique based GA [229]** (20): This method is specifically targeted to solve multiple sequence alignment problems encountered in bioinformatics. It is inspired by the behavior of an elastic rubber band on a plate with several poles. The poles can either be primary (through which the band ought to pass) or secondary. GA is used to arrive at the rubber band configuration which best addresses the sequence alignment problem.
56. **Bacterial-GA Foraging [230]** (19): It combines bacterial foraging algorithm with GA. In addition to tumble and swim, the bacteria (solutions) also undergo crossover and mutation as in a GA. When individuals are eliminated, new individuals are formed by mutating the eliminated ones instead of random reinitialization.
57. **Spiral Dynamics Inspired Optimization [231]** (19): The population consists candidate solutions, each forming the center of a spiral. The shape and size of the spirals is determined by two parameters, the rotation angle and the convergence rate to the center. These parameters control the diversification and intensification of the spiral around the solution. Spirals corresponding to high fitness centers tend towards intensification.
58. **Great Salmon Run Algorithm [232]** (16): The algorithm takes its inspiration from the salmon run phenomenon where millions of salmons migrate upstream for spawning even risking becoming prey. Each solution represents a salmon sub-group that selects a migration pathway. Different pathways pose different threats to the sub-group, like either being hunted by humans or by bears. At the end, salmons from safer pathways spawn to produce offspring which are more likely to take safer pathways.
59. **Paddy Field Algorithm [233]** (15): It uses the analogy with paddy fields where the seeds laid in the most fertile (high fitness) region of the field (search space) grow to produce more seeds. The seeds represent solutions and therefore eventually the optimum gets crowded.
60. **Japanese Tree Frogs Algorithm [234]** (13): The algorithm simulates the mating behavior of male tree frogs which intentionally desynchronize their mating calls to allow female frogs to correctly localize them. The analogy is used to solve the graph coloring problem, whose optimization version aims to minimize the number of colors to be used.
61. **Consultant-guided Search [235]** (13): Each solution in the population is a virtual person who can act both as a client and a consultant. The reputation of a person as a consultant depends on the number of successes achieved by its clients following the person's advice. Reputation can either increase or decrease, but when it falls below a certain value the person takes a sabbatical leave (reinitialization) and stops giving advice.
62. **Hierarchical Swarm Model [236]** (13): It generalizes the standard flat framework of swarm intelligence methods. Instead of all particles interacting with each other, this method introduces a

multiple level hierarchy. At each level the interaction is limited to the particles within the agents. Different agents interact at the next higher level. The topmost level consists of a single agent.

63. **Social Emotional Optimization [237]** (11): This algorithm is loosely based on the way humans modify their behavior to improve their social status. Based on an emotion index individuals (solutions) behave in a certain way. If this behavior is approved by the society (i.e. the solution has high fitness), the emotion index is increased so that this behavior becomes stronger.
64. **Anarchic Society Optimization [238]** (10): It is inspired by a hypothetical society whose members behave anarchically. The movement of solutions is based on the member's fickleness index, internal irregularity index and external irregularity index. Each of these represent the member's dissatisfaction with the current position, comparison with previous positions and affinity to others' positions.
65. **Eco-inspired evolutionary algorithm [239]** (10): It consists of multiple populations of individuals dispersed in the search space. Populations belonging to the same habitat are allowed to mate but those from different habitats are reproductively isolated. However, individuals may migrate to different habitats. While intra-habitat reproduction leads to intensification of solutions, inter -habitat migration leads to diversification. Both operations depend on the fitness landscape.
66. **OptBees Algorithm [240]** (9): Like the many other bee-based algorithms, it uses the concept of employer and onlooker bees which recruit more bees by dancing in relation to the best found food source (fitness). Scout bees are also used to search the decision space randomly.
67. **Termite Colony Optimization [241]** (9): The algorithm uses termites as a representation of solutions. A part of the termite population moves randomly and is responsible for providing diversity to the algorithm. The rest of the termites scan their neighborhood for pheromone content left by other termites and move towards the best location. If none is found, they perform a random walk.
68. **Egyptian Vulture Optimization [242]** (8): It mimics the nature, behavior and skills of Egyptian vultures for acquiring food. The algorithm primarily favors combinatorial optimization problems.
69. **Virtual Ant Algorithm [243]** (7): This is similar to ant colony optimization but is used to solve optimization problems other than traveling salesman problem. The virtual ants move thorough the search space and deposit pheromone whose levels are determined by the fitness of the region traversed by them.
70. **Bumblebees Multiagent Combinatorial optimization [244]** (5): The bumblebees algorithm was also developed to solve the graph coloring problem. A toroidal grid is used to represent solutions to this problem. The bumblebees' nest is randomly place in one of the cells and the bees are allowed to move to any other cell in search for food. Bumblebees that find food increase their lifespan by two units. After each generation the bumblebee's life is decreased by one unit and those which reach zero are removed.
71. **Atmosphere Clouds Model Optimization [245]** (3): In this method the search space is first divided into many disjoint regions. The humidity and air pressure in these regions is defined by the best known fitness value and the number of evaluated points in that region. Clouds form over a region when the humidity value is above a specified threshold. The movement of clouds to low pressure regions is equivalent to assigning more function evaluation to the lease explored regions. Water droplets from the clouds represent new solutions to be evaluated.
72. **Human-Inspired Algorithm [246]** (3): It is based on the way human climbers locate the highest peak by scanning the surroundings and communicating with fellow climbers. It divides the whole search space into equal subspaces and evenly assigns population members. The best fitness subspaces are allocated more members in the following generations by further dividing them into finer subspaces.
73. **Weightless Swarm Algorithm [247]** (2): It is a simplification of the conventional particle swarm optimization and has been used to solve dynamic optimization problems. The inertia weights are discarded and it also uses a modified update strategy.

74. **Good Lattice Swarm Algorithm [248]** (1): It replaces the random initialization of particles and their velocities in particle swarm optimization with a good lattice uniform distribution, which according to number theory, has the least discrepancy.
75. **Photosynthetic Algorithm [249]** (1): This algorithm uses the analogy with carbon molecules transforming from one substance to another during Calvin-Benson cycle and mimics the reaction taking place in the chloroplast during photorespiration for recombining different solutions.
76. **Eurygaster Algorithm [250]** (0): It mimics the behavior of eurygasters attacking grain farms in different groups always settling for the closest unattacked regions of the farm. The method has been applied to graph partitioning problem.

Many such methods are yet to be discovered and practiced, but an interesting future research would be to identify essential differences among these algorithms and specific features that make each of them unique and efficient in solving a specific class of optimization problems. Recent approaches [251, 252, 253] in this direction have provided useful insights about similarities of a few metaheuristic methodologies.

Despite their success and popularity, metaheuristic techniques have received some criticism over the years. The main criticism concerns their non-mathematical nature and lack of a proof of convergence. Due to the stochasticity involved, it is difficult to derive an exact theory of how these algorithms work. Few theoretical analyses do exist for some of the methods. However, they often relate to a simplified and basic version of the algorithm. While there is no immediate remedy for this, the best approach for practitioners is to provide empirical results on a wide variety of benchmark test problems available in literature. Reproducibility of results is an essential part of dissemination in this regard. In order to achieve this under stochasticity, uniformly seeded pseudo random number generators are used and the algorithm is run several times. A common practice is to use statistical analysis to draw conclusions from such empirical studies.

Recent criticism has been with respect to the ever growing number of "novel" metaheuristics as is evident from the list above. Sörensen [254] argues that many of these recently proposed metaheuristics are similar in almost all respects except the metaphor that is used to describe them. For example, in [255] and [256] the author shows that one of the recent metaheuristics, harmony search, is a special case of evolutionary strategy. Sörensen also points out that the terminology used to convey and justify the metaphor sometimes completely obfuscates the similarities with existing methods. Indeed, as described in Section 2, all metaheuristics have two major characteristics, exploration and exploitation. There may be several different ways of accomplishing these tasks in an algorithm, but calling each such method "new", "novel" and "innovative" would be pushing the metaphor too far. The major disadvantage of the undeserved attention that many of these recent techniques get is the distraction it causes away from key developments in the field of metaheuristics.

16. Conclusions

The present paper began with a discussion on the main concepts concerning metaheuristics such as their classification and characteristics. We laid down a generic framework into which most metaheuristic techniques can fit. The key components essential to the working of any metaheuristic were identified under five plans, namely selection, generation, replacement, update and termination. Some important parameters were also introduced. Thereafter, we reviewed a number of popular metaheuristic methods for solving optimization problems. Pseudocodes have been provided for 14 of the most commonly used methods for ease of understanding and implementation. Key variants of some of the algorithms were also discussed. Except simulated annealing and tabu search, all other methods are population-based. Moreover, we mainly concentrated on evolutionary and swarm-based methods, which are older, well-known and frequently used. For completeness, we have included an extensive list of other metaheuristics and ordered them by their Google Scholar citations. Though this metric may not be a true count of citations, it serves as a measure of popularity of these methods. Finally, we briefly discussed the criticism that the field of metaheuristics currently faces.

A recent trend in the field of metaheuristics is hybridization. This not only refers to the combination of different available metaheuristics but also to their integration with classical search methods. Classical algorithms are efficient local optimizers and therefore can aid the intensification process. Not surprisingly, such hybrid methods have been shown to outperform standalone methods. Hybrid-metaheuristics also refer to the use of data-mining and machine learning techniques to enhance search. With the increased accessibility to computing power, resource and memory-intensive tasks like data-mining can now be performed alongside metaheuristic optimization to learn patterns, predict outcomes and assist the search behavior.

Parallel metaheuristics can refer to different levels of parallelization, algorithmic level, iteration level and solution level. Algorithmic level involves multiple metaheuristics running simultaneously, communicating and collaborating with each other to solve a complex search problem. Iteration level involves parallelizing the operations within an iteration. This is especially useful in population-based metaheuristics. Solution level parallelization pertains to individual solutions. For example, if the evaluation of the objective function involves a finite element analysis, then this analysis itself can be parallelized. All these are very active and relevant areas of research within metaheuristics.

Another common research problem that needs to be addressed is that of setting the best parameters for the algorithm at hand. It is well-known that for most metaheuristics the performance can be drastically affected through their parameters. Parameter tuning refers to the selection of the best configuration of the algorithm for solving a particular problem. Both offline and online parameter tuning methods have been proposed in literature. However, like with optimization algorithms, there is no one best approach.

Metamodeling is an entire research area in itself. As metamodels are increasingly being used within optimization, more empirical studies need to be performed on the compatibility of different metamodels with different metaheuristics. Again, it is improbable that one unique combination will emerge as the winner. However, these studies can give a lot of insight about which methods work best on specific use-cases.

Metaheuristics have been and will continue to be used to solve complex optimization problems. Instead of developing more and more new algorithms, the focus should now shift towards improving current algorithms by unifying different ideas into a common and generic framework as the one described in this paper. There is also a growing need for classifying different metaheuristic methods according to their functional or algorithmic similarities. The algorithmic connection between two such methods provides a way to introduce one algorithm's key feature into another algorithm, a matter that will help unify different metaheuristic methods under one umbrella of a meta-algorithm for solving complex real-world optimization problems.

Acknowledgments

The first author acknowledges the financial support received from The Knowledge Foundation (KK-stiftelsen, Stockholm, Sweden) for the ProSpekt project KDISCO.

References

[1] E.-G. Talbi, *Metaheuristics: From design to implementation.* Hoboken, New Jersey, USA: John Wiley & Sons, 2009.
[2] G. Zäpfel, R. Braune, and M. Bögl, *Metaheuristic search concepts: A tutorial with applications to production and logistics.* Heidelberg: Springer Science & Business Media, 2010.
[3] M. Gendreau and J.-Y. Potvin, *Handbook of metaheuristics.* New York, USA: Springer, 2010.
[4] S. Luke, *Essentials of Metaheuristics.* Lulu, 2013. Available for free at http://cs.gmu.edu/∼sean/book/metaheuristics/.
[5] C. C. Ribeiro and P. Hansen, *Essays and surveys in metaheuristics.* New York, USA: Springer Science & Business Media, 2012.
[6] F. Glover and G. A. Kochenberger, *Handbook of metaheuristics.* Dordrecht: Kluwer Academic Publishers, 2003.
[7] I. H. Osman and J. P. Kelly, *Meta-heuristics: Theory and applications.* Norwell, Massachusetts, USA: Kluwer Academic Publishers, 2012.

[8] S. Voß, S. Martello, I. H. Osman, and C. Roucairol, *Meta-heuristics: Advances and trends in local search paradigms for optimization*. New York, USA: Springer Science & Business Media, 2012.
[9] T. F. Gonzalez, *Handbook of approximation algorithms and metaheuristics*. Boca Raton, FL, USA: CRC Press, 2007.
[10] J. Dréo, A. Petrowski, P. Siarry, and E. Taillard, *Metaheuristics for hard optimization: Methods and case studies*. Berlin Heidelberg: Springer Science & Business Media, 2006.
[11] P. Siarry and Z. Michalewicz, *Advances in metaheuristics for hard optimization*. Berlin Heidelberg: Springer Science & Business Media, 2007.
[12] K. F. Doerner, M. Gendreau, P. Greistorfer, W. Gutjahr, R. F. Hartl, and M. Reimann, *Metaheuristics: Progress in complex systems optimization*. New York, USA: Springer Science & Business Media, 2007.
[13] X.-S. Yang, *Nature-inspired optimization algorithms*. London, UK: Elsevier, 2014.
[14] D. F. Jones, S. K. Mirrazavi, and M. Tamiz, "Multi-objective meta-heuristics: An overview of the current state-of-the-art," *European Journal of Operational Research*, vol. 137, no. 1, pp. 1–9, 2002.
[15] I. H. Osman and G. Laporte, "Metaheuristics: A bibliography," *Annals of Operations Research*, vol. 63, no. 5, pp. 511–623, 1996.
[16] E.-G. Talbi, "A taxonomy of hybrid metaheuristics," *Journal of Heuristics*, vol. 8, no. 5, pp. 541–564, 2002.
[17] L. Jourdan, M. Basseur, and E.-G. Talbi, "Hybridizing exact methods and metaheuristics: A taxonomy," *European Journal of Operational Research*, vol. 199, no. 3, pp. 620–629, 2009.
[18] G. R. Raidl, "A unified view on hybrid metaheuristics," in *Hybrid Metaheuristics* (F. Almeida, M. J. B. Aguilera, C. Blum, J. M. M. Vega, M. P. Pérez, A. Roli, and M. Sampels, eds.), pp. 1–12, Springer, 2006.
[19] I. Boussaïd, J. Lepagnot, and P. Siarry, "A survey on optimization metaheuristics," *Information Sciences*, vol. 237, pp. 82–117, 2013.
[20] S. Olafsson, "Metaheuristics," in *Handbook in operations research and management science* (S. Henderson and B. Nelson, eds.), vol. 13, pp. 633–654, Elsevier, 2006.
[21] B. Melián, J. A. M. Pérez, and J. M. M. Vega, "Metaheuristics: A global view," *Inteligencia Artificial*, vol. 7, no. 19, pp. 7–28, 2003.
[22] C. Blum and A. Roli, "Hybrid metaheuristics: An introduction," in *Hybrid Metaheuristics* (C. Blum, M. J. B. Aguilera, A. Roli, and M. Sampels, eds.), pp. 1–30, Springer, 2008.
[23] C. Rego and F. Glover, "Local search and metaheuristics," in *The traveling salesman problem and its variations* (G. Gutin and A. P. Punnen, eds.), pp. 309–368, Springer, 2007.
[24] L. M. Rios and N. V. Sahinidis, "Derivative-free optimization: A review of algorithms and comparison of software implementations," *Journal of Global Optimization*, vol. 56, no. 3, pp. 1247–1293, 2013.
[25] C. Blum and A. Roli, "Metaheuristics in combinatorial optimization: Overview and conceptual comparison," *ACM Computing Surveys*, vol. 35, no. 3, pp. 268–308, 2003.
[26] L. Bianchi, M. Dorigo, L. M. Gambardella, and W. J. Gutjahr, "A survey on metaheuristics for stochastic combinatorial optimization," *Natural Computing*, vol. 8, no. 2, pp. 239–287, 2009.
[27] M. Gendreau and J.-Y. Potvin, "Metaheuristics in combinatorial optimization," *Annals of Operations Research*, vol. 140, no. 1, pp. 189–213, 2005.
[28] C. Blum, J. Puchinger, G. R. Raidl, and A. Roli, "Hybrid metaheuristics in combinatorial optimization: A survey," *Applied Soft Computing*, vol. 11, no. 6, pp. 4135–4151, 2011.
[29] T. G. Stützle, *Local search algorithms for combinatorial problems: Analysis, improvements, and new applications*. Amsterdam: IOS Press, 1999.
[30] L. Bianchi, M. Birattari, M. Chiarandini, M. Manfrin, M. Mastrolilli, L. Paquete, O. Rossi-Doria, and T. Schiavinotto, "Hybrid metaheuristics for the vehicle routing problem with stochastic demands," *Journal of Mathematical Modelling and Algorithms*, vol. 5, no. 1, pp. 91–110, 2006.
[31] O. Bräysy and M. Gendreau, "Vehicle routing problem with time windows, Part II: Metaheuristics," *Transportation Science*, vol. 39, no. 1, pp. 119–139, 2005.
[32] B. L. Golden, E. A. Wasil, J. P. Kelly, and I.-M. Chao, "The impact of metaheuristics on solving the vehicle routing problem: Algorithms, problem sets, and computational results," in *Fleet management and logistics* (T. G. Crainic and G. Laporte, eds.), pp. 33–56, Springer, 1998.
[33] W.-C. Chiang and R. A. Russell, "Simulated annealing metaheuristics for the vehicle routing problem with time windows," *Annals of Operations Research*, vol. 63, no. 1, pp. 3–27, 1996.
[34] J. Puchinger and G. R. Raidl, *Combining metaheuristics and exact algorithms in combinatorial optimization: A survey and classification*. Berlin Heidelberg: Springer, 2005.
[35] M. Birattari, L. Paquete, T. Stützle, and K. Varrentrapp, "Classification of metaheuristics and design of experiments for the analysis of components," tech. rep., Darmstadt University of Technology, Germany, 2001.
[36] D. J. Rosenkrantz, R. E. Stearns, and P. M. Lewis, II, "An analysis of several heuristics for the traveling salesman problem," *SIAM Journal on Computing*, vol. 6, no. 3, pp. 563–581, 1977.
[37] W. J. Gutjahr, "Convergence analysis of metaheuristics," in *Matheuristics* (V. Maniezzo, T. Stützle, and S. Voß, eds.), pp. 159–187, Springer, 2010.
[38] X.-S. Yang, "Metaheuristic optimization: Algorithm analysis and open problems," in *Experimental Algorithms* (P. M. Pardalos and S. Rebennack, eds.), pp. 21–32, Springer, 2011.
[39] T. Stützle and S. Fernandes, "New benchmark instances for the QAP and the experimental analysis of algorithms," in *Evolutionary Computation in Combinatorial Optimization* (J. Gottlieb and G. R. Raidl, eds.), pp. 199–209, Springer, 2004.

[40] M. Birattari, T. Stützle, L. Paquete, and K. Varrentrapp, "A racing algorithm for configuring metaheuristics," in *Proceedings of the Genetic and Evolutionary Computation Conference*, vol. 2, pp. 11–18, 2002.
[41] M. Birattari and M. Dorigo, *The problem of tuning metaheuristics as seen from a machine learning perspective*. PhD thesis, Universite Libre de Bruxelles, 2004.
[42] O. Rossi-Doria, M. Sampels, M. Birattari, M. Chiarandini, M. Dorigo, L. M. Gambardella, J. Knowles, M. Manfrin, M. Mastrolilli, B. Paechter, L. Paquete, and T. Stützle, "A comparison of the performance of different metaheuristics on the timetabling problem," in *Practice and Theory of Automated Timetabling IV* (E. Burke and P. D. Causmaecker, eds.), pp. 329–351, Springer, 2003.
[43] E. Alba, G. Luque, and E. Alba, "Measuring the performance of parallel metaheuristics," in *Parallel Metaheuristics: A New Class of Algorithms* (E. Alba, ed.), vol. 47, pp. 43–62, John Wiley & Sons, 2005.
[44] T. G. Crainic and M. Toulouse, *Parallel strategies for meta-heuristics*. New York, USA: Springer, 2003.
[45] E. Alba, *Parallel metaheuristics: A new class of algorithms*. Hoboken, New Jersey, USA: John Wiley & Sons, 2005.
[46] V.-D. Cung, S. L. Martins, C. C. Ribeiro, and C. Roucairol, "Strategies for the parallel implementation of metaheuristics," in *Essays and surveys in metaheuristics* (C. C. Ribeiro and P. Hansen, eds.), pp. 263–308, Springer, 2002.
[47] T. G. Crainic and M. Toulouse, "Parallel meta-heuristics," in *Handbook of metaheuristics* (M. Gendreau and J.-Y. Potvin, eds.), pp. 497–541, Springer, 2010.
[48] S. Cahon, N. Melab, and E.-G. Talbi, "Paradiseo: A framework for the reusable design of parallel and distributed metaheuristics," *Journal of Heuristics*, vol. 10, no. 3, pp. 357–380, 2004.
[49] E.-G. Talbi, *Parallel combinatorial optimization*. Hoboken, New Jersey, USA: John Wiley & Sons, 2006.
[50] E. Alba, G. Luque, and S. Nesmachnow, "Parallel metaheuristics: Recent advances and new trends," *International Transactions in Operational Research*, vol. 20, no. 1, pp. 1–48, 2013.
[51] E. Alba and G. Luque, "Evaluation of parallel metaheuristics," in *Parallel Problem Solving from Nature*, pp. 9–14, Springer, 2006.
[52] S. D. Ekşioğlu, P. Pardalos, and M. Resende, "Parallel metaheuristics for combinatorial optimization," in *Models for Parallel and Distributed Computation* (R. Corrêa, I. Dutra, M. Fiallos, and F. Gomes, eds.), Springer, 2002.
[53] D. Wolpert and W. Macready, "No free lunch theorems for optimization," *IEEE Transactions on Evolutionary Computation*, vol. 1, no. 1, pp. 67–82, 1997.
[54] Z. Michalewicz, "A survey of constraint handling techniques in evolutionary computation methods," *Evolutionary Programming*, vol. 4, pp. 135–155, 1995.
[55] C. A. C. Coello, "Theoretical and numerical constraint-handling techniques used with evolutionary algorithms: A survey of the state of the art," *Computer methods in applied mechanics and engineering*, vol. 191, no. 11, pp. 1245–1287, 2002.
[56] K. Deb and R. Datta, "A fast and accurate solution of constrained optimization problems using a hybrid bi-objective and penalty function approach," in *IEEE Congress on Evolutionary Computation*, pp. 165–172, IEEE, 2010.
[57] S. Camazine, J.-L. Deneubourg, N. R. Franks, J. Sneyd, G. Theraulaz, and E. Bonabeau, *Self-organization in biological systems*. Princeton, New Jersey, USA: Princeton University Press, 2003.
[58] P. Grassé, "Recherches sur la biologie des termites champignonnistes (Macrotermitinae)," *Annales des Sciences Naturelles (Zoologie)*, vol. 6, pp. 97–171, 1944.
[59] J.-L. Deneubourg, S. Aron, S. Goss, and J. M. Pasteels, "The self-organizing exploratory pattern of the argentine ant," *Journal of Insect Behavior*, vol. 3, no. 2, pp. 159–168, 1990.
[60] M. Dorigo, V. Maniezzo, and A. Colorni, "Ant system: Optimization by a colony of cooperating agents," *IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics*, vol. 26, no. 1, pp. 29–41, 1996.
[61] E. L. Lawler, J. K. Lenstra, A. H. G. R. Kan, and D. B. Shmoys, *The traveling salesman problem: A guided tour of combinatorial optimization*. Chichester, UK: Wiley, 1985.
[62] V. Maniezzo and A. Colorni, "The ant system applied to the quadratic assignment problem," *IEEE Transactions on Knowledge and Data Engineering*, vol. 11, no. 5, pp. 769–778, 1999.
[63] A. Colorni, M. Dorigo, V. Maniezzo, and M. Trubian, "Ant system for job-shop scheduling," *Belgian Journal of Operations Research, Statistics and Computer Science*, vol. 34, no. 1, pp. 39–53, 1994.
[64] B. Bullnheimer, R. F. Hartl, and C. Strauss, "Applying the ant system to the vehicle routing problem," in *Meta-Heuristics* (S. Voß, S. Martello, I. H. Osman, and C. Roucairol, eds.), pp. 285–296, Springer, 1999.
[65] M. Dorigo and L. M. Gambardella, "Ant colony system: A cooperative learning approach to the traveling salesman problem," *IEEE Transactions on Evolutionary Computation*, vol. 1, no. 1, pp. 53–66, 1997.
[66] G. Bilchev and I. C. Parmee, "The ant colony metaphor for searching continuous design spaces," in *Evolutionary Computing* (T. C. Fogarty, ed.), pp. 25–39, Springer, 1995.
[67] K. Socha and M. Dorigo, "Ant colony optimization for continuous domains," *European Journal of Operational Research*, vol. 185, no. 3, pp. 1155–1173, 2008.
[68] M. Mathur, S. B. Karale, S. Priye, V. Jayaraman, and B. Kulkarni, "Ant colony approach to continuous function optimization," *Industrial & engineering chemistry research*, vol. 39, no. 10, pp. 3814–3822, 2000.
[69] G. Crina and A. Ajith, "Stigmergic optimization: Inspiration, technologies and perspectives," in *Stigmergic optimization* (A. Ajith, G. Crina, and R. Vitorino, eds.), pp. 1–24, Springer, 2006.
[70] D. Karaboga, "An idea based on honey bee swarm for numerical optimization," tech. rep., Erciyes University, Turkey, 2005.
[71] D. Karaboga and B. Basturk, "A powerful and efficient algorithm for numerical function optimization: Artificial bee

colony (ABC) algorithm," *Journal of Global Optimization*, vol. 39, no. 3, pp. 459–471, 2007.
[72] R. S. Rao, S. Narasimham, and M. Ramalingaraju, "Optimization of distribution network configuration for loss reduction using artificial bee colony algorithm," *International Journal of Electrical Power and Energy Systems Engineering*, vol. 1, no. 2, pp. 116–122, 2008.
[73] A. Singh, "An artificial bee colony algorithm for the leaf-constrained minimum spanning tree problem," *Applied Soft Computing*, vol. 9, no. 2, pp. 625–631, 2009.
[74] N. Karaboga, "A new design method based on artificial bee colony algorithm for digital iir filters," *Journal of the Franklin Institute*, vol. 346, no. 4, pp. 328–348, 2009.
[75] Q.-K. Pan, M. F. Tasgetiren, P. N. Suganthan, and T. J. Chua, "A discrete artificial bee colony algorithm for the lot-streaming flow shop scheduling problem," *Information Sciences*, vol. 181, no. 12, pp. 2455–2468, 2011.
[76] D. Teodorović and M. Dell'Orco, "Bee colony optimization - A cooperative learning approach to complex transportation problems," in *Proceedings of the 16th Mini-EURO Conference on Advanced OR and AI Methods in Transportation*, pp. 51–60, 2005.
[77] S. Omkar, J. Senthilnath, R. Khandelwal, G. N. Naik, and S. Gopalakrishnan, "Artificial bee colony (ABC) for multi-objective design optimization of composite structures," *Applied Soft Computing*, vol. 11, no. 1, pp. 489–499, 2011.
[78] S. F. M. Burnet, *The clonal selection theory of acquired immunity.* Cambridge, United Kingdom: Cambridge University Press, 1959.
[79] L. N. De Castro and F. J. Von Zuben, "The clonal selection algorithm with engineering applications," in *Proceedings of the Genetic and Evolutionary Computation Conference*, vol. 2000, pp. 36–39, 2000.
[80] L. N. De Castro and F. J. Von Zuben, "Learning and optimization using the clonal selection principle," *IEEE Transactions on Evolutionary Computation*, vol. 6, no. 3, pp. 239–251, 2002.
[81] J. Timmis, C. Edmonds, and J. Kelsey, "Assessing the performance of two immune inspired algorithms and a hybrid genetic algorithm for function optimisation," in *IEEE Congress on Evolutionary Computation*, vol. 1, pp. 1044–1051, IEEE, 2004.
[82] A. Gasper and P. Collard, "From GAs to artificial immune systems: Improving adaptation in time dependent optimization," in *IEEE Congress on Evolutionary Computation*, vol. 3, IEEE, 1999.
[83] L. N. De Castro and F. J. Von Zuben, "Learning and optimization using the clonal selection principle," *IEEE Transactions on Evolutionary Computation*, vol. 6, no. 3, pp. 239–251, 2002.
[84] L. N. De Castro and F. J. Von Zuben, "The clonal selection algorithm with engineering applications," in *Proceedings of the Genetic and Evolutionary Computation Conference*, vol. 2000, pp. 36–39, 2000.
[85] P. Hajela and J. Lee, "Constrained genetic search via schema adaptation: An immune network solution," *Structural Optimization*, vol. 12, no. 1, pp. 11–15, 1996.
[86] X. Wang, X. Z. Gao, and S. J. Ovaska, "An immune-based ant colony algorithm for static and dynamic optimization," in *International Conference on Systems, Man and Cybernetics*, pp. 1249–1255, IEEE, 2007.
[87] D. Dasgupta, *An overview of artificial immune systems and their applications.* Berlin Heidelberg: Springer, 1999.
[88] R. Storn and K. Price, "Differential evolution - A simple and efficient heuristic for global optimization over continuous spaces," *Journal of Global Optimization*, vol. 11, no. 4, pp. 341–359, 1997.
[89] J. A. Nelder and R. Mead, "A simplex method for function minimization," *The Computer Journal*, vol. 7, no. 4, pp. 308–313, 1965.
[90] S. Das and P. N. Suganthan, "Differential evolution: A survey of the state-of-the-art," *IEEE Transactions on Evolutionary Computation*, vol. 15, no. 1, pp. 4–31, 2011.
[91] R. Storn and K. Price, "Minimizing the real functions of the icec'96 contest by differential evolution," in *IEEE International Conference on Evolutionary Computation*, pp. 842–844, IEEE, 1996.
[92] K. V. Price, R. M. Storn, and J. A. Lampinen, *Differential evolution - A practical approach to global optimization.* Berlin Heidelberg: Springer-Verlag, 2005.
[93] V. Feoktistov and S. Janaqi, "Generalization of the strategies in differential evolution," in *Proceedings of the 18th International Parallel and Distributed Processing Symposium*, pp. 2341–2346, IEEE, 2004.
[94] H.-Y. Fan and J. Lampinen, "A trigonometric mutation operation to differential evolution," *Journal of Global Optimization*, vol. 27, no. 1, pp. 105–129, 2003.
[95] K. V. Price, "An introduction to differential evolution," in *New ideas in optimization* (D. Corne, M. Dorigo, F. Glover, D. Dasgupta, P. Moscato, R. Poli, and K. V. Price, eds.), pp. 79–108, McGraw-Hill, 1999.
[96] H.-P. Schwefel, "Kybernetische evolution als strategie der experimentellen forschung in der strömungstechnik," Master's thesis, Technical University of Berlin, 1965.
[97] I. Rechenberg, *Evolutionsstrategien.* Berlin Heidelberg: Springer, 1978.
[98] I. Rechenberg, *Evolutionsstrategie: Optimierung technischer Systeme nach Prinzipien der biologishen Evolution.* Stuttgart: Frommann-Holzboog Verlag, 1973.
[99] H.-G. Beyer and H.-P. Schwefel, "Evolution strategies - A comprehensive introduction," *Natural Computing*, vol. 1, no. 1, pp. 3–52, 2002.
[100] H.-P. Schwefel, *Evolutionsstrategie und numerische Optimierung.* PhD thesis, Technical University of Berlin, 1975.
[101] H.-P. Schwefel, *Numerische Optimierung von Computer-Modellen mittels der Evolutionsstrategie: Mit einer vergleichenden Einführung in die Hill-Climbing-und Zufallsstrategie.* Basel: Birkhäuser Verlag, 1977.
[102] T. Back, F. Hoffmeister, and H.-P. Schwefel, "A survey of evolution strategies," in *Proceedings of the Fourth International Conference on Genetic Algorithms*, pp. 2–9, Morgan Kaufmann, 1991.

[103] H.-G. Beyer, *The theory of evolution strategies*. Berlin: Springer, 2001.
[104] A. Auger and N. Hansen, "Theory of evolution strategies: A new perspective," *Theory of Randomized Search Heuristics: Foundations and Recent Developments*, vol. 1, pp. 289–325, 2011.
[105] H.-G. Beyer, "Toward a theory of evolution strategies: Self-adaptation," *Evolutionary Computation*, vol. 3, no. 3, pp. 311–347, 1995.
[106] H.-G. Beyer and K. Deb, "On self-adaptive features in real-parameter evolutionary algorithms," *IEEE Transactions on Evolutionary Computation*, vol. 5, no. 3, pp. 250–270, 2001.
[107] N. Hansen and A. Ostermeier, "Completely derandomized self-adaptation in evolution strategies," *Evolutionary Computation*, vol. 9, no. 2, pp. 159–195, 2001.
[108] N. Hansen, S. D. Müller, and P. Koumoutsakos, "Reducing the time complexity of the derandomized evolution strategy with covariance matrix adaptation (CMA-ES)," *Evolutionary Computation*, vol. 11, no. 1, pp. 1–18, 2003.
[109] N. Hansen and S. Kern, "Evaluating the CMA evolution strategy on multimodal test functions," in *Parallel Problem Solving from Nature*, pp. 282–291, Springer, 2004.
[110] A. Auger and N. Hansen, "A restart CMA evolution strategy with increasing population size," in *IEEE Congress on Evolutionary Computation*, vol. 2, pp. 1769–1776, IEEE, 2005.
[111] N. Hansen and A. Ostermeier, "Adapting arbitrary normal mutation distributions in evolution strategies: The covariance matrix adaptation," in *IEEE International Conference on Evolutionary Computation*, pp. 312–317, IEEE, 1996.
[112] L. J. Fogel, A. J. Owens, and M. J. Walsh, *Artificial intelligence through simulated evolution*. New York, USA: John Wiley, 1966.
[113] D. B. Fogel, "An introduction to simulated evolutionary optimization," *IEEE Transactions on Neural Networks*, vol. 5, no. 1, pp. 3–14, 1994.
[114] T. Bäck and H.-P. Schwefel, "An overview of evolutionary algorithms for parameter optimization," *Evolutionary Computation*, vol. 1, no. 1, pp. 1–23, 1993.
[115] D. B. Fogel, "An analysis of evolutionary programming," in *Proceedings of the First Annual Conference on Evolutionary Programming*, pp. 43–51, 1992.
[116] D. B. Fogel, L. J. Fogel, and J. W. Atmar, "Meta-evolutionary programming," in *Conference on Signals, systems and computers*, pp. 540–545, IEEE, 1991.
[117] X. Yao, Y. Liu, and G. Lin, "Evolutionary programming made faster," *IEEE Transactions on Evolutionary Computation*, vol. 3, no. 2, pp. 82–102, 1999.
[118] D. B. Fogel, *Evolving Artificial Intelligence*. PhD thesis, University of California, San Diego, 1992.
[119] T. Bäck, G. Rudolph, and H.-P. Schwefel, "Evolutionary programming and evolution strategies: Similarities and differences," in *Proceedings of the Second Annual Conference on Evolutionary Programming*, pp. 11–22, Evolutionary Programming Society, 1993.
[120] J. Holland, *Adaptation in natural and artificial systems*. Cambridge, Massachusetts, USA: MIT Press, 1992.
[121] D. Goldberg, *Genetic algorithms in search, optimization, and machine learning*. Reading, Massachusetts, USA: Addison-Wesley, 1989.
[122] F. Herrera, M. Lozano, and J. Verdegay, "Tackling real-coded genetic algorithms: Operators and tools for behavioural analysis," *Artificial Intelligence Review*, vol. 12, no. 4, pp. 265–319, 1998.
[123] L. Eshelman and J. Schaffer, "Real-coded genetic algorithms and interval-schemata," in *Proceedings of the Second Workshop on Foundations of Genetic Algorithms*, vol. 2, pp. 187–202, Morgan Kaufmann, 1992.
[124] K. Deb and R. B. Agrawal, "Simulated binary crossover for continuous search space," *Complex Systems*, vol. 9, no. 3, pp. 1–15, 1994.
[125] H.-M. Voigt, H. Mühlenbein, and D. Cvetkovic, "Fuzzy recombination for the breeder genetic algorithm," in *Proceedings of the Sixth International Conference on Genetic Algorithms*, pp. 104–113, Morgan Kaufmann, 1995.
[126] Z. Michalewicz, *Genetic algorithms + Data structures = Evolution programs*. Berlin Heidelberg: Springer, 1996.
[127] H.-P. Schwefel, "Collective phenomena in evolutionary systems," in *Problems of Constancy and Change - The Complementarity of Systems Approaches to Complexity*, pp. 1025–1033, International Society for General Systems Research, 1987.
[128] D. Fogel, *Evolutionary computation: Toward a new philosophy of machine intelligence*. Hoboken, New Jersey, USA: Wiley-IEEE Press, 2006.
[129] K. Deb and M. Goyal, "A combined genetic adaptive search (GeneAS) for engineering design," *Computer Science and Informatics*, vol. 26, no. 4, pp. 30–45, 1996.
[130] G. Rudolph, "Convergence analysis of canonical genetic algorithms," *IEEE Transactions on Neural Networks*, vol. 5, no. 1, pp. 96–101, 1994.
[131] D. E. Goldberg and J. Richardson, "Genetic algorithms with sharing for multimodal function optimization," in *Proceedings of the Second International Conference on Genetic Algorithms*, pp. 41–49, Hillsdale, 1987.
[132] A. Pétrowski, "A clearing procedure as a niching method for genetic algorithms," in *IEEE International Conference on Evolutionary Computation*, pp. 798–803, 1996.
[133] L. J. Eshelman, "The CHC adaptive search algorithm: How to have safe search when engaging in non-traditional genetic recombination," in *Foundations of Genetic Algorithms*, Morgan Kaufmann, 1990.
[134] J. Joines and C. R. Houck, "On the use of non-stationary penalty functions to solve nonlinear constrained optimization problems with ga's," in *IEEE Congress on Evolutionary Computation*, pp. 579–584, IEEE, 1994.
[135] D. Whitley, V. S. Gordon, and K. Mathias, "Lamarckian evolution, the baldwin effect and function optimization," in

Parallel Problem Solving from Nature (Y. Davidor, H.-P. Schwefel, and R. Männer, eds.), pp. 5–15, Springer, 1994.
[136] J. R. Koza, *Genetic Programming: On the programming of computers by means of natural selection*. Cambridge, Massachusetts, USA: MIT Press, 1992.
[137] M. F. Brameier and W. Banzhaf, *Linear genetic programming*. New York, USA: Springer, 2007.
[138] R. Poli, "Evolution of graph-like programs with parallel distributed genetic programming," in *Proceedings of the Seventh International Conference on Genetic Algorithms*, pp. 346–353, Morgan Kaufmann, 1997.
[139] J. F. Miller and P. Thomson, "Cartesian genetic programming," in *Genetic Programming* (R. Poli, W. Banzhaf, W. B. Langdon, J. Miller, P. Nordin, and T. C. Fogarty, eds.), pp. 121–132, Springer, 2000.
[140] R. Poli, W. Langdon, N. McPhee, and J. Koza, "Genetic programming: An introductory tutorial and a survey of techniques and applications," tech. rep., University of Essex, UK, 2007.
[141] J. R. Koza, *Genetic programming II: Automatic discovery of reusable programs*. Cambridge, Massachusetts, USA: MIT Press, 1994.
[142] J. R. Koza, F. H. Bennett III, and O. Stiffelman, *Genetic programming as a Darwinian invention machine*. Berlin Heidelberg: Springer, 1999.
[143] P. J. Angeline and J. B. Pollack, "The evolutionary induction of subroutines," in *Proceedings of the fourteenth annual conference of the cognitive science society*, pp. 236–241, 1992.
[144] J. P. Rosca and D. H. Ballard, "Discovery of subroutines in genetic programming," in *Advances in Genetic Programming 2* (P. J. Angeline and J. K. E. Kinnear, eds.), pp. 177–202, MIT Press, 1996.
[145] P. Nordin and W. Banzhaf, "Complexity compression and evolution," in *Proceedings of the Sixth International Conference on Genetic Algorithms*, pp. 310–317, 1995.
[146] B.-T. Zhang and H. Mühlenbein, "Balancing accuracy and parsimony in genetic programming," *Evolutionary Computation*, vol. 3, no. 1, pp. 17–38, 1995.
[147] S. Luke and L. Panait, "A comparison of bloat control methods for genetic programming," *Evolutionary Computation*, vol. 14, no. 3, pp. 309–344, 2006.
[148] W. Banzhaf, P. Nordin, R. E. Keller, and F. D. Francone, *Genetic Programming: An Introduction*. San Francisco, CA, USA: Morgan Kaufmann, 1997.
[149] M. Clerc, *Particle swarm optimization*. London, UK: ISTE Ltd., 2010.
[150] Y. Shi and R. C. Eberhart, "Parameter selection in particle swarm optimization," in *Proceedings of the Seventh International Conference on Evolutionary Programming*, pp. 591–600, Springer, 1998.
[151] R. Eberhart, P. Simpson, and R. Dobbins, *Computational Intelligence PC Tools*. San Diego, CA, USA: Academic Press Professional, 1996.
[152] Y. Shi and R. Eberhart, "A modified particle swarm optimizer," in *IEEE Congress on Evolutionary Computation*, pp. 69–73, IEEE, 1998.
[153] A. Ratnaweera, S. K. Halgamuge, and H. C. Watson, "Self-organizing hierarchical particle swarm optimizer with time-varying acceleration coefficients," *IEEE Transactions on Evolutionary Computation*, vol. 8, no. 3, pp. 240–255, 2004.
[154] P. Tawdross and A. König, "Local parameters particle swarm optimization," in *International Conference on Hybrid Intelligent Systems*, pp. 52–52, IEEE, 2006.
[155] M. Clerc and J. Kennedy, "The particle swarm-explosion, stability, and convergence in a multidimensional complex space," *IEEE Transactions on Evolutionary Computation*, vol. 6, no. 1, pp. 58–73, 2002.
[156] F. Glover, "Heuristics for integer programming using surrogate constraints," *Decision Sciences*, vol. 8, no. 1, pp. 156–166, 1977.
[157] F. Glover, "Scatter search and path relinking," in *New ideas in optimization* (D. Corne, M. Dorigo, F. Glover, D. Dasgupta, P. Moscato, R. Poli, and K. V. Price, eds.), McGraw-Hill, 1999.
[158] F. Glover, M. Laguna, and R. Martí, "Fundamentals of scatter search and path relinking," *Control and cybernetics*, vol. 29, no. 3, pp. 653–684, 2000.
[159] M. Laguna and R. Marti, *Scatter search: Methodology and implementations in C*. New York, USA: Springer Science & Business Media, 2012.
[160] N. Metropolis, A. W. Rosenbluth, M. N. Rosenbluth, A. H. Teller, and E. Teller, "Equation of state calculations by fast computing machines," *The Journal of Chemical Physics*, vol. 21, no. 6, pp. 1087–1092, 1953.
[161] F. Glover, "Future paths for integer programming and links to artificial intelligence," *Computers & Operations Research*, vol. 13, no. 5, pp. 533–549, 1986.
[162] F. Glover and E. Taillard, "A user's guide to tabu search," *Annals of Operations Research*, vol. 41, no. 1, pp. 1–28, 1993.
[163] F. Glover, "Tabu search - Part I," *ORSA Journal on Computing*, vol. 1, no. 3, pp. 190–206, 1989.
[164] F. Glover, "Tabu search - Part II," *ORSA Journal on Computing*, vol. 2, no. 1, pp. 4–32, 1990.
[165] M. Gendreau, A. Hertz, and G. Laporte, "A tabu search heuristic for the vehicle routing problem," *Management Science*, vol. 40, no. 10, pp. 1276–1290, 1994.
[166] F. Glover and M. Laguna, *Tabu Search*. New York, USA: Springer, 2013.
[167] W. G. Zong, *Music-inspired Harmony Search Algorithm: Theory and Applications*. Berlin Heidelberg: Springer, 2009.
[168] Z. W. Geem, J. H. Kim, and G. Loganathan, "A new heuristic optimization algorithm: Harmony search," *Simulation*, vol. 76, no. 2, pp. 60–68, 2001.
[169] P. Larrañaga and J. A. Lozano, *Estimation of distribution algorithms: A new tool for evolutionary computation*. New

York, USA: Springer, 2002.
[170] S. Baluja, "Population-based incremental learning: A method for integrating genetic search based function optimization and competitive learning," tech. rep., Carnegie Mellon University, USA, 1994.
[171] K. M. Passino, "Biomimicry of bacterial foraging for distributed optimization and control," *IEEE Control Systems Magazine*, vol. 22, no. 3, pp. 52–67, 2002.
[172] P. Moscato, "On evolution, search, optimization, genetic algorithms and martial arts: Towards memetic algorithms," tech. rep., Caltech Concurrent Computation Program, California Institute of Technology, USA, 1989.
[173] C. Ferreira, "Gene expression programming: A new adaptive algorithm for solving problems," *Complex Systems*, vol. 13, no. 2, pp. 87–129, 2001.
[174] W. D. Hillis, "Co-evolving parasites improve simulated evolution as an optimization procedure," *Physica D: Nonlinear Phenomena*, vol. 42, no. 1, pp. 228–234, 1990.
[175] M. A. Potter and K. A. De Jong, "A cooperative coevolutionary approach to function optimization," in *Parallel Problem Solving from Nature* (Y. Davidor, H.-P. Schwefel, and R. Männer, eds.), pp. 249–257, Springer, 1994.
[176] X.-S. Yang, *Nature-inspired metaheuristic algorithms*. Frome, UK: Luniver Press, 2010.
[177] X.-S. Yang, "Firefly algorithms for multimodal optimization," in *Stochastic algorithms: Foundations and applications* (O. Watanabe and T. Zeugmann, eds.), pp. 169–178, Springer, 2009.
[178] E. Rashedi, H. Nezamabadi-Pour, and S. Saryazdi, "GSA: A gravitational search algorithm," *Information Sciences*, vol. 179, no. 13, pp. 2232–2248, 2009.
[179] X.-S. Yang and S. Deb, "Cuckoo search via Lévy flights," in *World Congress on Nature & Biologically Inspired Computing*, pp. 210–214, IEEE, 2009.
[180] D. Simon, "Biogeography-based optimization," *IEEE Transactions on Evolutionary Computation*, vol. 12, no. 6, pp. 702–713, 2008.
[181] M. Eusuff, K. Lansey, and F. Pasha, "Shuffled frog-leaping algorithm: A memetic meta-heuristic for discrete optimization," *Engineering Optimization*, vol. 38, no. 2, pp. 129–154, 2006.
[182] M. M. Eusuff and K. E. Lansey, "Optimization of water distribution network design using the shuffled frog leaping algorithm," *Journal of Water Resources Planning and Management*, vol. 129, no. 3, pp. 210–225, 2003.
[183] D. Pham, A. Ghanbarzadeh, E. Koc, S. Otri, S. Rahim, and M. Zaidi, "The bees algorithm - A novel tool for complex optimisation problems," in *International Conference on Intelligent Production Machines and Systems*, pp. 454–459, 2006.
[184] R. G. Reynolds, "An introduction to cultural algorithms," in *Proceedings of the third annual conference on evolutionary programming*, pp. 131–139, World Scientific, 1994.
[185] E. Atashpaz-Gargari and C. Lucas, "Imperialist competitive algorithm: An algorithm for optimization inspired by imperialistic competition," in *IEEE Congress on Evolutionary Computation*, pp. 4661–4667, IEEE, 2007.
[186] B. Xing and W.-J. Gao, "Imperialist competitive algorithm," in *Innovative Computational Intelligence: A Rough Guide to 134 Clever Algorithms* (B. Xing and W.-J. Gao, eds.), pp. 203–209, Springer, 2014.
[187] X.-S. Yang, "A new metaheuristic bat-inspired algorithm," in *Nature Inspired Cooperative Strategies for Optimization* (J. R. González, D. A. Pelta, C. Cruz, G. Terrazas, and N. Krasnogor, eds.), pp. 65–74, Springer, 2010.
[188] C. Ryan, J. Collins, and M. O. Neill, "Grammatical evolution: Evolving programs for an arbitrary language," in *Genetic Programming* (W. Banzhaf, R. Poli, M. Schoenauer, and T. C. Fogarty, eds.), pp. 83–96, Springer, 1998.
[189] M. O'Neill and C. Ryan, "Grammatical evolution," *IEEE Transactions on Evolutionary Computation*, vol. 5, no. 4, pp. 349–358, 2001.
[190] A. R. Mehrabian and C. Lucas, "A novel numerical optimization algorithm inspired from weed colonization," *Ecological Informatics*, vol. 1, no. 4, pp. 355–366, 2006.
[191] A. Kaveh and S. Talatahari, "A novel heuristic optimization method: Charged system search," *Acta Mechanica*, vol. 213, no. 3-4, pp. 267–289, 2010.
[192] O. K. Erol and I. Eksin, "A new optimization method: Big bang–big crunch," *Advances in Engineering Software*, vol. 37, no. 2, pp. 106–111, 2006.
[193] H. Abbass, "MBO: Marriage in honey bees optimization - A haplometrosis polygynous swarming approach," in *IEEE Congress on Evolutionary Computation*, vol. 1, pp. 207–214, IEEE, 2001.
[194] D. Teodorović and M. Dell'Orco, "Bee colony optimization - A cooperative learning approach to complex transportation problems," in *Advanced OR and AI Methods in Transportation*, pp. 51–60, 2005.
[195] S. He, Q. H. Wu, and J. Saunders, "Group search optimizer: An optimization algorithm inspired by animal searching behavior," *IEEE Transactions on Evolutionary Computation*, vol. 13, no. 5, pp. 973–990, 2009.
[196] A. Afshar, O. Bozorg Haddad, M. A. Mariño, and B. Adams, "Honey-bee mating optimization (HBMO) algorithm for optimal reservoir operation," *Journal of the Franklin Institute*, vol. 344, no. 5, pp. 452–462, 2007.
[197] A. H. Gandomi and A. H. Alavi, "Krill herd: A new bio-inspired optimization algorithm," *Communications in Nonlinear Science and Numerical Simulation*, vol. 17, no. 12, pp. 4831–4845, 2012.
[198] H. F. Wedde, M. Farooq, and Y. Zhang, "Beehive: An efficient fault-tolerant routing algorithm inspired by honey bee behavior," in *Ant colony optimization and swarm intelligence* (M. Dorigo, M. Birattari, C. Blum, L. M. Gambardella, F. Mondada, and T. Stützle, eds.), pp. 83–94, Springer, 2004.
[199] K. Krishnanand and D. Ghose, "Detection of multiple source locations using a glowworm metaphor with applications to collective robotics," in *IEEE Swarm Intelligence Symposium*, pp. 84–91, IEEE, 2005.
[200] H. Shah-Hosseini, "The intelligent water drops algorithm: A nature-inspired swarm-based optimization algorithm," *International Journal of Bio-Inspired Computation*, vol. 1, no. 1-2, pp. 71–79, 2009.

[201] P. Lucic and D. Teodorovic, "Bee system: Modeling combinatorial optimization transportation engineering problems by swarm intelligence," in *Preprints of the TRISTAN IV triennial symposium on transportation analysis*, pp. 441–445, 2001.
[202] J. Bishop, "Stochastic searching networks," in *First IEE International Conference on Artificial Neural Networks*, pp. 329–331, IET, 1989.
[203] S.-C. Chu, P.-W. Tsai, and J.-S. Pan, "Cat swarm optimization," in *PRICAI 2006: Trends in Artificial Intelligence* (Q. Yang and G. Webb, eds.), pp. 854–858, Springer, 2006.
[204] R. A. Formato, "Central force optimization: A new metaheuristic with applications in applied electromagnetics," *Progress In Electromagnetics Research*, vol. 77, pp. 425–491, 2007.
[205] S. H. Jung, "Queen-bee evolution for genetic algorithms," *Electronics Letters*, vol. 39, no. 6, pp. 575–576, 2003.
[206] H. Drias, S. Sadeg, and S. Yahi, "Cooperative bees swarm for solving the maximum weighted satisfiability problem," in *Computational Intelligence and Bioinspired Systems* (J. Cabestany, A. Prieto, and F. Sandoval, eds.), pp. 318–325, Springer, 2005.
[207] X.-L. Li and J.-X. Qian, "Studies on artificial fish swarm optimization algorithm based on decomposition and coordination techniques," *Journal of Circuits and Systems*, vol. 1, pp. 1–6, 2003.
[208] X.-S. Yang and S. Deb, "Eagle strategy using Lévy walk and firefly algorithms for stochastic optimization," in *Nature Inspired Cooperative Strategies for Optimization* (J. R. González, D. A. Pelta, C. Cruz, G. Terrazas, and N. Krasnogor, eds.), pp. 101–111, Springer, 2010.
[209] P. Civicioglu, "Transforming geocentric cartesian coordinates to geodetic coordinates by using differential search algorithm," *Computers & Geosciences*, vol. 46, pp. 229–247, 2012.
[210] P. Civicioglu, "Backtracking search optimization algorithm for numerical optimization problems," *Applied Mathematics and Computation*, vol. 219, no. 15, pp. 8121–8144, 2013.
[211] X.-S. Yang, "Flower pollination algorithm for global optimization," in *Unconventional Computation and Natural Computation* (J. Durand-Lose and N. Jonoska, eds.), pp. 240–249, Springer, 2012.
[212] I. Zelinka, "Analytic programming by means of SOMA algorithm," in *Proceedings of the Eighth International Conference on Soft Computing*, vol. 2, pp. 93–101, 2002.
[213] J. Carmelo Filho, F. B. De Lima Neto, A. J. Lins, A. I. Nascimento, and M. P. Lima, "A novel search algorithm based on fish school behavior," in *IEEE International Conference on Systems, Man and Cybernetics*, pp. 2646–2651, IEEE, 2008.
[214] C. J. Bastos Filho, F. B. de Lima Neto, A. J. Lins, A. I. Nascimento, and M. P. Lima, "Fish school search," in *Nature-Inspired Algorithms for Optimisation* (R. Chiong, ed.), pp. 261–277, Springer, 2009.
[215] A. Hatamlou, "Black hole: A new heuristic optimization approach for data clustering," *Information Sciences*, vol. 222, pp. 175–184, 2013.
[216] A. Mucherino and O. Seref, "Monkey search: A novel metaheuristic search for global optimization," in *Data Mining, Systems Analysis and Optimization in Biomedicine* (O. Seref, O. E. Kundakcioglu, and P. Pardalos, eds.), vol. 953, pp. 162–173, AIP Publishing, 2007.
[217] Y. Chu, H. Mi, H. Liao, Z. Ji, and Q. Wu, "A fast bacterial swarming algorithm for high-dimensional function optimization," in *IEEE Congress on Evolutionary Computation*, pp. 3135–3140, IEEE, 2008.
[218] H. Eskandar, A. Sadollah, A. Bahreininejad, and M. Hamdi, "Water cycle algorithm - A novel metaheuristic optimization method for solving constrained engineering optimization problems," *Computers & Structures*, vol. 110-111, pp. 151–166, 2012.
[219] G. Kjellstrom and L. Taxen, "Stochastic optimization in system design," *IEEE Transactions on Circuits and Systems*, vol. 28, no. 7, pp. 702–715, 1981.
[220] P. Rabanal, I. Rodríguez, and F. Rubio, "Using river formation dynamics to design heuristic algorithms," in *Unconventional Computation* (S. G. Akl, C. S. Calude, M. J. Dinneen, G. Rozenberg, and H. T. Wareham, eds.), pp. 163–177, Springer, 2007.
[221] Y. Shi, "An optimization algorithm based on brainstorming process," *International Journal of Swarm Intelligence Research*, vol. 2, no. 4, pp. 35–62, 2011.
[222] T. C. Havens, C. J. Spain, N. G. Salmon, and J. M. Keller, "Roach infestation optimization," in *Swarm Intelligence Symposium*, pp. 1–7, IEEE, 2008.
[223] A. H. Kashan, "League championship algorithm: A new algorithm for numerical function optimization," in *International Conference of Soft Computing and Pattern Recognition*, pp. 43–48, IEEE, 2009.
[224] E. Cuevas, D. Oliva, D. Zaldivar, M. Pérez-Cisneros, and H. Sossa, "Circle detection using electro-magnetism optimization," *Information Sciences*, vol. 182, no. 1, pp. 40–55, 2012.
[225] H. Shah-Hosseini, "Principal components analysis by the galaxy-based search algorithm: A novel metaheuristic for continuous optimisation," *International Journal of Computational Science and Engineering*, vol. 6, no. 1, pp. 132–140, 2011.
[226] A. Kaveh and N. Farhoudi, "A new optimization method: Dolphin echolocation," *Advances in Engineering Software*, vol. 59, pp. 53–70, 2013.
[227] R. Tang, S. Fong, X.-S. Yang, and S. Deb, "Wolf search algorithm with ephemeral memory," in *IEEE Seventh International Conference on Digital Information Management*, 2012.
[228] P. Civicioglu, "Artificial cooperative search algorithm for numerical optimization problems," *Information Sciences*, vol. 229, pp. 58–76, 2012.
[229] J. Taheri and A. Y. Zomaya, "RBT-GA: A novel metaheuristic for solving the multiple sequence alignment problem,"

BMC Genomics, vol. 10, no. 1, p. Article ID S10, 2009.
[230] T.-C. Chen, P.-W. Tsai, S.-C. Chu, and J.-S. Pan, "A novel optimization approach: Bacterial-GA foraging," in *International Conference on Innovative Computing, Information and Control*, pp. 391–391, IEEE, 2007.
[231] K. Tamura and K. Yasuda, "Spiral dynamics inspired optimization," *Journal of Advanced Computational Intelligence and Intelligent Informatics*, vol. 15, no. 8, pp. 1116–1122, 2011.
[232] A. Mozaffari, A. Fathi, and S. Behzadipour, "The great salmon run: A novel bio–inspired algorithm for artificial system design and optimisation," *International Journal of Bio-Inspired Computation*, vol. 4, no. 5, pp. 286–301, 2012.
[233] U. Premaratne, J. Samarabandu, and T. Sidhu, "A new biologically inspired optimization algorithm," in *International Conference on Industrial and Information Systems*, pp. 279–284, IEEE, 2009.
[234] H. Hernández and C. Blum, "Distributed graph coloring: An approach based on the calling behavior of japanese tree frogs," *Swarm Intelligence*, vol. 6, no. 2, pp. 117–150, 2012.
[235] S. Iordache, "Consultant-guided search: A new metaheuristic for combinatorial optimization problems," in *Proceedings of the Genetic and Evolutionary Computation Conference*, pp. 225–232, ACM, 2010.
[236] H. Chen, Y. Zhu, K. Hu, and X. He, "Hierarchical swarm model: A new approach to optimization," *Discrete Dynamics in Nature and Society*, vol. 2010, p. Article ID 379649, 2010.
[237] Y. Xu, Z. Cui, and J. Zeng, "Social emotional optimization algorithm for nonlinear constrained optimization problems," in *Swarm, Evolutionary, and Memetic Computing* (B. K. Panigrahi, S. Das, P. N. Suganthan, and S. S. Dash, eds.), pp. 583–590, Springer, 2010.
[238] H. Shayeghi and J. Dadashpour, "Anarchic society optimization based PID control of an automatic voltage regulator (AVR) system," *Electrical and Electronic Engineering*, vol. 2, no. 4, pp. 199–207, 2012.
[239] R. S. Parpinelli and H. S. Lopes, "An eco-inspired evolutionary algorithm applied to numerical optimization," in *World Congress on Nature & Biologically Inspired Computing*, pp. 466–471, IEEE, 2011.
[240] R. D. Maia, L. N. de Castro, and W. M. Caminhas, "Bee colonies as model for multimodal continuous optimization: The OptBees algorithm," in *IEEE Congress on Evolutionary Computation*, pp. 1–8, IEEE, 2012.
[241] R. Hedayatzadeh, F. Akhavan Salmassi, M. Keshtgari, R. Akbari, and K. Ziarati, "Termite colony optimization: A novel approach for optimizing continuous problems," in *18th Iranian Conference on Electrical Engineering*, pp. 553–558, IEEE, 2010.
[242] C. Sur, S. Sharma, and A. Shukla, "Egyptian vulture optimization algorithm - A new nature inspired meta-heuristics for knapsack problem," in *International Conference on Computing and Information Technology*, pp. 227–237, Springer, 2013.
[243] X.-S. Yang, J. M. Lees, and C. T. Morley, "Application of virtual ant algorithms in the optimization of CFRP shear strengthened precracked structures," in *Proceedings of the Sixth International Conference on Computational Science*, pp. 834–837, Springer, 2006.
[244] F. Comellas and J. Martinez-Navarro, "Bumblebees: A multiagent combinatorial optimization algorithm inspired by social insect behaviour," in *Proceedings of the first ACM/SIGEVO Summit on Genetic and Evolutionary Computation*, pp. 811–814, ACM, 2009.
[245] G.-W. Yan and Z.-J. Hao, "A novel optimization algorithm based on atmosphere clouds model," *International Journal of Computational Intelligence and Applications*, vol. 12, no. 1, p. Article ID 1350002, 2013.
[246] L. M. Zhang, C. Dahlmann, and Y. Zhang, "Human-inspired algorithms for continuous function optimization," in *International Conference on Intelligent Computing and Intelligent Systems*, vol. 1, pp. 318–321, IEEE, 2009.
[247] T. Ting, K. L. Man, S.-U. Guan, M. Nayel, and K. Wan, "Weightless swarm algorithm (WSA) for dynamic optimization problems," in *Network and Parallel Computing* (J. J. Park, A. Zomaya, S.-S. Yeo, and S. Sahni, eds.), pp. 508–515, Springer, 2012.
[248] S. Su, J. Wang, W. Fan, and X. Yin, "Good lattice swarm algorithm for constrained engineering design optimization," in *International Conference on Wireless Communications, Networking and Mobile Computing*, pp. 6421–6424, IEEE, 2007.
[249] B. Alatas, "Photosynthetic algorithm approaches for bioinformatics," *Expert Systems with Applications*, vol. 38, no. 8, pp. 10541–10546, 2011.
[250] F. Ahmadi, H. Salehi, and K. Karimi, "Eurygaster algorithm: A new approach to optimization," *International Journal of Computer Applications*, vol. 57, no. 2, pp. 9–13, 2012.
[251] K. A. DeJong, *Evolutionary computation: A unified approach*. Cambridge, Massachusetts, USA: MIT Press, 2006.
[252] K. Deb and N. Padhye, "Development of efficient particle swarm optimizers by using concepts from evolutionary algorithms," in *Proceedings of the Genetic and Evolutionary Computation Conference*, pp. 55–62, ACM, 2010.
[253] N. Padhye, P. Bhardawaj, and K. Deb, "Improving differential evolution through a unified approach," *Journal of Global Optimization*, vol. 55, no. 4, pp. 771–799, 2013.
[254] K. Sörensen, "Metaheuristics - The metaphor exposed," *International Transactions in Operational Research*, vol. 22, no. 1, pp. 3–18, 2015.
[255] D. Weyland, "A rigorous analysis of the harmony search algorithm: How the research community can be misled by a "novel" methodology," *International Journal of Applied Metaheuristic Computing*, vol. 1, no. 2, pp. 50–60, 2010.
[256] D. Weyland, "A critical analysis of the harmony search algorithm - How not to solve Sudoku," *Operations Research Perspectives*, vol. 2, pp. 97–105, 2015.