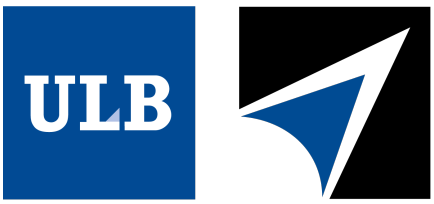


ECOLE
POLYTECHNIQUE
DE BRUXELLES

Designing new metaheuristic implementations:
From metaphors to automatic design

Thesis presented by Christian L. CAMACHO VILLALÓN
in fulfilment of the requirements of the
PhD Degree in Engineering Sciences and Technology
("Docteur en Sciences de l'ingénieur et technologie")
Année académique 2022-2023

Supervisor: Professor Marco DORIGO
Co-supervisor: Professor Thomas STÜTZLE
IRIDIA - Institut de Recherches Interdisciplinaires et de
Développements en Intelligence Artificielle

Designing new metaheuristic implementations: From metaphors to automatic design

Christian L. Camacho Villalón

IRIDIA, Université libre de Bruxelles, Belgium.

2023

Composition of the jury

Prof. Mauro BIRATTARI

Research Director of the F.R.S.-FNRS at IRIDIA, École polytechnique de Bruxelles, Université libre de Bruxelles, Belgium.

Prof. Marco DORIGO (supervisor)

Research Director of the F.R.S.-FNRS and co-director of IRIDIA, École polytechnique de Bruxelles, Université libre de Bruxelles, Belgium.

Prof. Leslie PÉREZ CÁCERES

Associate Professor, Escuela de Ingeniería Informática, Pontificia Universidad Católica de Valparaíso, Chile.

Prof. Kenneth SÖRENSEN

Full Professor, Department of Engineering Management, University of Antwerp, Belgium.

Prof. Thomas STÜTZLE (co-supervisor)

Research Director of the F.R.S.-FNRS at IRIDIA, École polytechnique de Bruxelles, Université libre de Bruxelles, Belgium.

Abstract

A metaheuristic is a general algorithmic framework that can be easily adapted to many different optimization problems for which exact/analytical approaches are either limited or impractical. There are two main approaches used to create new metaheuristic implementations: *manual design*, which is based on the designer's "intuition"—often involving looking for inspiration in other fields of knowledge—and *automatic design*, which seeks to remove human intervention from the design process by harnessing recent advances in automatic algorithm configuration methods and machine learning. Compared to manual design, which is typically time-consuming and error-prone, automatic design is often much more efficient and has advantages, such as removing biases from the creation process and increasing the flexibility with which new metaheuristic designs can be explored. Yet, to this day, the vast majority of metaheuristics proposed in the literature are still created by algorithm designers who manually define the different components that make up the implementation based on their knowledge (empirical, theoretical or intuitive) and/or by considering new sources of inspiration.

The main topic of this thesis is how to improve the way we create metaheuristics by replacing manual design with automatic design. We focus especially on those cases where the use of manual design involves taking inspiration from natural and even supernatural behaviors, since this approach has been used intensively in the last two decades, spawning hundreds of so-called "novel" metaphor-based metaheuristics. Unfortunately, for the vast majority of them, it is still unknown what is their real novelty apart from the use of the new metaphors. In order to put an end to this situation and, indeed, to try to remedy it, we advocate for replacing "intuition" and "new sources of inspiration" with automated design methods. To demonstrate the feasibility of this approach and its advantages, we develop a metaheuristic software framework with a modular design, that is coupled with an automatic configuration tool and allows to automatically create high-performing implementations with novel designs.

In the first half of this doctoral thesis, we focus on studying how the process of designing metaheuristics has changed over the years. In doing so, we identify the limitations of the approach of looking for inspiration in other fields of knowledge and establish clear criteria to determine the cases where considering introducing new metaphors makes *sense* and those where it does not. In this endeavor, we rigorously analyze some of the most widespread and highly-cited "novel" metaphor-based metaheuristics proposed in the last years, and compare their components with those defined in some of the best-known metaheuristics that have been proposed in the literature. We show that, despite being presented as *novel* optimization techniques, they are in fact the same as, or at best minor variations of, classic approaches, many of which were proposed years, or even decades, before the "novel" metaheuristic were published.

In the other half of this thesis, we explore the use of automatic design as a powerful alternative to manual design, that has also the potential of rendering the need to find new sources of inspiration obsolete. We describe how modular metaheuristic software frameworks are created, what are their advantages and challenges, and why they are the most efficient way we have at the moment to try to come up with new metaheuristic designs. Successfully, we experimentally demonstrate that the metaheuristic software framework for particle swarm optimization that we developed in the context of this research work can be used to create high-performing implementations whose design had never been considered before in the literature, without the need for introducing new sources of inspiration, or any other kind of manual intervention. This doctoral thesis concludes by: (i) identifying ways in which some of the fundamental aspect of the broad field of metaheuristics can be improved; (ii) discussing the research contributions presented in this work; and (iii) identifying some research paths that can be explored in the future to give continuation to this work.

Declaration of Authorship

This thesis presents an original work that has never been submitted to the Université libre de Bruxelles or to any other institution for the award of a doctoral degree. Some parts of this thesis are based on a number of peer-reviewed articles that the author, together with his supervisor, co-supervisor and other collaborators, has published in the scientific literature.

Parts of the introduction (Chapter 1), background (Chapters 2 and 4) and discussion (Chapter 9) are based on:

- **Camacho-Villalón, C.L**, Stützle, T., and Dorigo, M. (2023). "Designing New Metaheuristics: Manual versus Automated Approaches". In: *Intelligent Computing, A Science Partner Journal*. Accepted for publication.
- **Camacho-Villalón, C.L.**, Dorigo, M., and Stützle, T. (2022b). "Exposing the grey wolf, moth-flame, whale, firefly, bat, and antlion algorithms: six misleading optimization techniques inspired by bestial metaphors". In: *International Transactions in Operational Research* 30.6, pp. 2945–2971.

The comparison between manual and automatic approaches (Chapter 3), and the description of metaheuristic software frameworks (Chapter 7) are based on:

- **Camacho-Villalón, C.L**, Stützle, T., and Dorigo, M. (2023). "Designing New Metaheuristics: Manual versus Automated Approaches". In: *Intelligent Computing, A Science Partner Journal*. Accepted for publication.

The analysis of the "novel" metaheuristics for discrete optimization (Chapter 5) is based on:

- **Camacho-Villalón, C.L.**, Dorigo, M., and Stützle, T. (2018). "Why the Intelligent Water Drops Cannot Be Considered as a Novel Algorithm". In: *Swarm Intelligence, 11th International Conference, ANTS 2018*. Vol. 11172. Lecture Notes in Computer Science. Springer, pp. 302–314.

- **Camacho-Villalón, C.L.**, Dorigo, M., and Stützle, T. (2019). "The intelligent water drops algorithm: why it cannot be considered a novel algorithm". In: *Swarm Intelligence* 13.3–4, pp. 173–192.

The analysis of the "novel" metaheuristics for continuous optimization (Chapter 6) is based on:

- **Camacho-Villalón, C.L**, Stützle, T., and Dorigo, M. (2020). "Grey Wolf, Firefly and Bat Algorithms: Three Widespread Algorithms that Do Not Contain Any Novelty". In: *Swarm Intelligence, 12th International Conference, ANTS 2020*. Vol. 12421. Lecture Notes in Computer Science. Springer, pp. 121–133.

- **Camacho-Villalón, C.L.**, Dorigo, M., and Stützle, T. (2022a). "An analysis of why cuckoo search does not bring any novel ideas to optimization". In: *Computers & Operations Research* 142, p. 105747.

- **Camacho-Villalón, C.L.**, Dorigo, M., and Stützle, T. (2022b). "Exposing the grey wolf, moth-flame, whale, firefly, bat, and antlion algorithms: six misleading optimization techniques inspired by bestial metaphors". In: *International Transactions in Operational Research* 30.6, pp. 2945–2971.

The metheuristic software framework for particle swarm optimization (Chapter 8) is based on:

- **Camacho-Villalón, C.L.**, Dorigo, M., and Stützle, T. (2022c). "PSO-X: A Component-Based Framework for the Automatic Design of Particle Swarm Optimization Algorithms". In: *IEEE Transactions on Evolutionary Computation* 26.3, pp. 402–416.

Acknowledgements

The successful completion of this Ph.D. thesis marks the end of a journey that I began several years ago. I would like to express my deepest gratitude to all those who have contributed, directly or indirectly, to this once-in-a-lifetime experience. Your guidance, support and friendship have left an indelible mark on my academic and personal growth.

First and foremost, I would like to extend my heartfelt appreciation to my amazing supervisors, Marco and Thomas: thank you both for being the best supervisors one could have wish for. Your knowledge and patience have been instrumental in my research. You have also guided me towards the right path every time, and I have learned a lot during this process. I am truly grateful for the opportunity to work under your mentorship.

I would like to thank the members of my thesis committee, Prof. Mauro Birattari, Prof. Leslie Pérez Cáceres and Prof. Kenneth Sörensen, for their valuable insights and constructive feedback during the evaluation of this research. They have immensely contributed to the refinement of this work, and I am truly grateful for their time and efforts.

I would like to acknowledge financial support from the Combinatorial Optimization: Metaheuristics & Exact Methods (COMEX) project, the National Council of Science and Technology (CONACyT), in Mexico, and the F.R.S.-FNRS (via an Aspirant fellowship No. 34824958). Their investment in my research has enabled me to carry out this work and has been crucial in its successful completion. I am grateful for their belief in the importance of my research and for their commitment to advancing knowledge in the field of metaheuristics.

My sincere appreciation goes to the other professors of IRIDIA, Hugues and Mauro. You contributed immensely to make IRIDIA an amazing lab with an incredibly nice atmosphere. A great thanks also to all the staff of the Université Libre de Bruxelles for their dedication to providing a conducive academic environment, research resources, and technical support.

I would also like to thank my fellow IRIDIANs who have provided constant support and encouragement throughout this amazing journey. Their discussions, suggestions, and moments of camaraderie have been a source of inspiration and motivation. I am grateful for the uncountable stimulating conversations we had during this time. I am sure we will continue developing what already feels like a lifelong friendship. A special mention goes to my friend and colleague, Alberto Franzin, who has been a mentor during my time as a doctoral student: thanks Alberto for always making time for me to discuss my work.

My deepest gratitude goes to my parents, Blanca and Carlos, for their unwavering love, encouragement, and constant support. Their belief in my abilities and their sacrifices have been the driving force behind my accomplishments. I am forever grateful for their faith in me.

Finally, I would like to give special thanks to Claudia, my wonderful and loving girlfriend, who has been my companion in pretty much everything I have done during all this time. Claudia: you have been an amazing partner; you have helped me to solve many challenging problems; you have taught me to see the positive side of things; and you have given me support, encouragement, and helpful advice on my work. I could not have had a better companion during this time of my life.

— Christian —

Contents

Chapter 1

Introduction

Optimization is a vast research field with hundreds of years of history that deals with a large variety of optimization problems and solution methods. Although the early days of optimization were characterized by the development of algorithms that could find optimal solutions, it eventually became clear that many optimization problems cannot be efficiently solved to optimality. Well-known examples of such problems are multimodal non-differentiable functions in the continuous optimization domain (Andréasson et al. 2020; Luenberger, Ye, et al. 2016), and NP-hard problems in the discrete optimization domain (Garey and Johnson 1979; Papadimitriou and Steiglitz 1982; Tovey 2002). With the advent of ever more powerful computers, heuristic algorithms rapidly became the mainstream approach to deal with difficult optimization problems, replacing in many cases the use of exact algorithms. In other words, the focus of the research moved from the design and development of algorithms that find the *best* solution, to the design and development of algorithms that can rapidly provide solutions that are very good, although not provably optimal. Since the seminal work by Glover (1986), the most used term to refer to this type of algorithms is *metaheuristic*.

There have been many attempts to provide a definition of the term "metaheuristic" that is at the same time precise and allows to include all the diverse metaheuristics that have been proposed in the literature. The definition provided by the Metaheuristics Network (*Metaheuristics Network. Project Summary* n.d.), which is the one we adopt in this work, is:

> "A metaheuristic is a set of algorithmic concepts that can be used to define heuristic methods applicable to a wide set of different problems. In other words, a metaheuristic can be seen as a general

> algorithmic framework which can be applied to different optimization problems with relatively few modifications to make it adapted to a specific problem."

Some of the most popular and best performing metaheuristics currently available in the literature include evolutionary computation (Fogel et al. 1966; Holland 1975; Rechenberg 1973; Schwefel 1977, 1981), tabu search (Glover 1989, 1990), simulated annealing (Černý 1985; Kirkpatrick 1984), ant colony optimization (Dorigo 1992a; Dorigo et al. 1991b; Dorigo and Stützle 2004), particle swarm optimization (Eberhart and Kennedy 1995; Kennedy et al. 2001; Kennedy and Eberhart 1995) and iterated local search (Ramalhinho Lourenço et al. 2002).

Over the years, the research on the different metaheuristics accumulated into a plethora of different metaheuristic implementations in which the concepts defining the metaheuristic (also called metaheuristic components) were implemented in many different ways, with the goal of improving the efficiency of the corresponding metaheuristic implementations. Researchers started then to create new metaheuristic implementations that were no longer just proposing new ways of implementing the metaheuristic components, but that were also reusing components proposed in previous metaheuristic implementations. This greatly increased the number of design choices to be made when creating a new metaheuristic implementation. Thus, it became less and less efficient to approach the creation of metaheuristic implementations manually—i.e., by *handcrafting* one by one the different components that make up the metaheuristic implementation.

In order to devise new metaheuristic designs, and motivated by the early successes of metaheuristics that were inspired by natural processes, such as evolutionary computation, simulated annealing, and ant colony optimization, a fraction of the metaheuristics community has been proposing "novel" metaheuristics based on a disparate set of metaphors. Unfortunately, not only this approach is just another instance of the inefficient manual design approach, but it has also caused a number of undesirable consequences for the entire field. Among them, one of the most negative is that, more often than not, the only novelty in a proposed "novel" metaheuristic seems to be the use of new—and often confusing—terminology that hides its similarities with already published metaheuristics and algorithms (Aranha et al. 2022; Armas et al. 2022; Campelo and Aranha 2021a; Lones 2020; Piotrowski et al. 2014; Sörensen 2015; Thymianis and Tzanetos 2022; Tzanetos and Dounias 2021; Weyland 2010).

The initial motivation for this research is the need to better understand the problem of the so-called "novel" metaphor-based algorithms and, in particular, the causes that led to the intensive use of an approach that often seems to lack any scientific motivation. To do so, in Chapter 2, we begin by studying some of the classic metaphor-based metaheuristics published in the literature. For each of them, we illustrate how the natural metaphor was of paramount importance to their development, identify the algorithm concepts that have been proposed in them, and explain how these concepts are used to perform optimization. In Chapter 3, we discuss the two main approaches to design metaheuristics and highlight their upsides and downsides. Based on all this necessary background, in Chapter 4, we examine in great detail the problem of the "novel" metaphor-based metaheuristic and explain why these "novel" metaheuristics are so problematic, what are the negative consequences they have created, and what efforts have been conducted so far by the scientific community to address them.

Another important motivation for this research is the need to clarify the real novelty of the hundreds of "novel" metaphor-based metaheuristics proposed in the literature. Therefore, a substantial part of this work has been devoted to perform rigorous, component-based analyses of these metaheuristics. We present these analyses in Chapters 5 and Chapters 6. Our findings are similar to those found by other rigorous analyses (see, e.g., (Piotrowski et al. 2014; Weyland 2010)): the studied "novel" metaheuristics can be exactly mapped to already published techniques just by rephrasing the terminology used to describe them (Camacho-Villalón et al. 2018, 2019, 2022a, 2023, 2020). Even though these analyses are just a few compared to the number of "novel" metaheuristics already published in the literature (which is around 500), their contribution to the research community is relevant in our opinion, as they present concrete instances of the problem and help to raise awareness about their negative consequences.

Whereas Chapters 2 – 6 aim at providing the reader with a clear account of the problem that is tackled in this work, Chapters 7 and 8 are dedicated to propose possible solutions. In particular, we consider the use of automatic algorithm design methods as a way to avoid altogether the approach of looking for inspiration in other fields of knowledge to try to come up with new metaheuristic designs. Automatic design, whose fundamentals are given in Chapter 3, is a relatively recent approach in which manual human intervention is increasingly less important (Stützle and López-Ibáñez 2019) and the need for

novel metaphors disappears. The development of these automated methods and their application to create efficient metaheuristic implementations is nowadays a central topic in the field of metaheuristics.

In Chapter 7, we describe how the automatic design approach is being used to create the *new* generation of high-performing metaheuristic implementations. Then, in Chapter 8, we present PSO-*X*, a metaheuristic software framework for particle swarm optimization that was developed in the context of this research and that includes a large number of components proposed for this metaheuristic that cover more than 20 years of its research and developments. The motivation for proposing PSO-*X* goes beyond putting together many different variants proposed for particle swarm optimization in one single place. We also aimed at providing the metaheuristics community with a tool from which it was possible to automatically instantiate a number of metaheuristic implementations that, according to our analyses, are variants of particle swarm optimization. We discuss the potential of PSO-*X* to do this at the end of Chapter 8.

Finally, in Chapter 9, we reflect on some of the fundamental aspect of the broad field of metaheuristics that we believe should be re-thought in order to continue pushing the field forward. This chapter elaborates on why we should (i) focus on experimentally- or theoretically-driven research rather than on purely application-driven research and competitive testing; (ii) use state-of-the-art benchmarking practices to evaluate metaheuristics; and (iii) use modern tools and mechanisms to automatically create high-performing metaheuristic implementations. In Chapter 10, we present the conclusions of this thesis and identify future research directions.

1.1 Preview of Contributions

The following is a summary of the contributions presented in this thesis:

A detailed review of the way metaheuristics are designed: We provide a detailed review of the two main approaches used to design metaheuristics (i.e., manual design and automatic design) in which we contrast their benefits and challenges.

Perspective for the field of metaheuristics: We present a critical perspective for the field that discusses fundamental aspect that can be improved in order to solve the issue of the "novel" metaheuristics and continue pushing the field forward.

- **Camacho-Villalón, Christian Leonardo**, Stützle, Thomas, and Dorigo, Marco (2023). "Designing New Metaheuristics: Manual versus Automatic Approaches". In: *Intelligent Computing, A Science Partner Journal*. Accepted for publication.

Rigorous analyses of the most popular "novel" metaphor-based metaheuristics: We present a rigorous, component-by-component analysis of some of the most popular and highly-cited "novel" metaphor-based metaheuristics. In these analyses, we identify the ideas proposed in the "novel" metaheuristics and compared them with those propose in other well-established techniques proposed before in the literature in order to highlight their real novelty.

- **Camacho-Villalón, Christian Leonardo**, Dorigo, Marco, and Stützle, Thomas (2018). "Why the Intelligent Water Drops Cannot Be Considered as a Novel Algorithm". In: *Swarm Intelligence, 11th International Conference, ANTS 2018*. Vol. 11172. Lecture Notes in Computer Science. Springer, pp. 302–314.
- **Camacho-Villalón, Christian Leonardo**, Dorigo, Marco, and Stützle, Thomas (2019). "The intelligent water drops algorithm: why it cannot be considered a novel algorithm". In: *Swarm Intelligence* 13.3–4, pp. 173–192.
- **Camacho-Villalón, Christian Leonardo**, Stützle, Thomas, and Dorigo, Marco (2020). "Grey Wolf, Firefly and Bat Algorithms: Three Widespread Algorithms that Do Not Contain Any Novelty". In: *Swarm Intelligence, 12th International Conference, ANTS 2020*. Vol. 12421. Lecture Notes in Computer Science. Springer, pp. 121–133.
- **Camacho-Villalón, Christian Leonardo**, Dorigo, Marco, and Stützle, Thomas (2022a). "An analysis of why cuckoo search does not bring any novel ideas to optimization". In: *Computers & Operations Research* 142, p. 105747.
- **Camacho-Villalón, Christian Leonardo**, Dorigo, Marco, and Stützle, Thomas (2022b). "Exposing the grey wolf, moth-flame, whale, firefly, bat, and antlion algorithms: six misleading optimization techniques inspired by bestial metaphors". In: *International Transactions in Operational Research* 30.6, pp. 2945–2971.

Open letter about metaphor-based metaheuristics: Together with other like-minded scientist, we published an open letter that describes some of the unscientific practices used in the field of metaheuristics and proposes a number of

necessary actions to address them. This open letter was signed by more than 90 scientist and practitioners working on the field of metaheuristics.

- Aranha, Claus, **Camacho-Villalón**, **Christian Leonardo**, Campelo, Felipe, Dorigo, Marco, Ruiz, Rubén, Sevaux, Marc, Sörensen, Kenneth, and Stützle, Thomas (2022). "Metaphor-based Metaheuristics, a Call for Action: the Elephant in the Room". In: *Swarm Intelligence* 16.1, pp. 1–6.

PSO-*X*: We introduce PSO-*X*, a novel modular, automatically configurable metaheuristic framework that includes a repertoires of components proposed particle swarm optimization for over 20 years. This framework allows to automatically generate high-performing implementation without the need of human intervention and, by extension, without having to introduce new metaphors.

- **Camacho-Villalón, Christian Leonardo**, Dorigo, Marco, and Stützle, Thomas (2022c). "PSO-X: A Component-Based Framework for the Automatic Design of Particle Swarm Optimization Algorithms". In: *IEEE Transactions on Evolutionary Computation* 26.3, pp. 402–416.

Part I

Background

Chapter 2

Optimization Problems and Metaheuristics

2.1 Optimization Problems

Optimization problems arise in all domains and fields, and consist in finding solutions that are optimal or near-optimal with respect to some goal (Rothlauf 2011). To give a few examples, we can consider someone with a limited budget trying to get as many items as possible on a shopping list, an e-commerce company selecting the routes for a fleet of vehicles serving customers in different geographical zones, or an engineer designing the chassis for a new car. In the optimization literature, these problems are known, respectively, as the "knapsack problem" (Garey and Johnson 1979), the "vehicle routing problem" (Dantzig and Ramser 1959), and as an "engineering design problem" (Deb 2012). Even though we may not think much of it, being able to solve increasingly difficult optimization problems has been instrumental in virtually all areas of life, such as technology, manufacturing, supply chain, health care, finance, education, commerce, etc. In addition to their practical relevance and complexity, that we discuss below, the initial distinction that can be made between optimization problems is based on the domain of their solution variables, which can be *discrete*, *continuous*, or a mix of the two.

In combinatorial optimization problems, the goal is to find groupings, orderings or assignments of a discrete, finite set of values that satisfy some given conditions. In formal terms, a combinatorial optimization problem can be modeled as a tuple $\Pi = (S, \Omega, f)$, where:

- S is the search space (also called the decision space) of the problem, and it

is defined by a finite set of n decision variables, $x_1, \ldots, x_n$, each with its own domain, typically a subset of $\mathbb{Z}$;

- Ω is a set of constraints among the decision variables; and
- $f : S \to \mathbb{R}$ is an objective function that maps S to $\mathbb{R}$.

A feasible solution to a combinatorial optimization problem is given by assigning a possible value to each variable x_i in S, for $i = 1, \ldots, n$, that satisfies the constraints in Ω.

Many combinatorial optimization problems are NP-hard, which means that the computational effort required to solve them grows exponentially with the size of the problem instances (Garey and Johnson 1979; Papadimitriou and Steiglitz 1982). Some well-known examples of NP-hard combinatorial optimization problems are: the traveling salesman problems, which involve finding shortest tours on graphs; the Boolean satisfiability problems, which involve finding models of propositional formulae; and job-scheduling problems, which involve creating schedules for jobs to be processed by a number of machines.

On the other hand, in continuous optimization problems, the goal is to find real-valued vectors that minimize/maximize a continuous function. Without loss of generality, we consider minimization problems, where the goal is to minimize a d-dimensional continuous objective function $f : S \subseteq \mathbb{R}^d \to \mathbb{R}$ by finding a vector $\vec{o} \in S$, such that $\forall\, \vec{x} \in S$, $f(\vec{o}) \leq f(\vec{x})$. The search space S is a subset of $\mathbb{R}^d$ in which a solution is represented by a real-valued vector $\vec{x}$, and each component x^j of $\vec{x}$ is constrained by a lower and upper bound such that $lb^j \leq x^j \leq ub^j$, for $j = 1, \ldots, d$. The vector $\vec{o}$ represents the solution for which the evaluation function $f(\cdot)$ returns the minimum value. For a maximization problem, the obvious adaptation consists in using $-f(\cdot)$ instead of $f(\cdot)$.

Finding optimal solutions to continuous optimization problems can be very difficult, if not impossible, when the problems are not well-structured (i.e., when the objective function is not explicitly known) or when there are constraints on their decision variables (Luenberger, Ye, et al. 2016; Rothlauf 2011). Some classical examples of optimization problems in continuous domains that are difficult to solve are engineering design problems (such as airfoils, springs, vessels, etc.), financial problems (such as portfolio selection and index tracking problems), and parameter optimization problems in black-box scenarios[1] (such

[1]Optimization problems in black-box scenarios are those in which the objective function of

as managing power grids and developing personalized health-care treatments).

Over the years, many different methods have been developed to solve optimization problems. Some of these methods are *exact*, i.e., they guarantee to find the optimal solution; while others are *heuristic* or *metaheuristic*, i.e., they are intended to find good solutions in short computational times without offering any guarantees about their optimality. While both exact and (meta)heuristic approaches have advantages and disadvantages, the decision of which method to use typically depends on the complexity of the optimization problem.

For example, if the considered problem turns out to be NP-hard, an exact algorithm such as *branch and bound* will not be able to solve a large instance of the problem to optimality because the time taken by this method to find such an optimal solution will grow exponentially with the size of the problem instance. A similar scenario is obtained when dealing with continuous, non-differentiable, multimodal functions, although, in this case, the problem is that applying analytical/numerical methods, such as calculating the *Hessian matrix* of the objective function or applying *gradient search*, requires a large computational effort and it is limited to small areas of the search space where the optimal solution is believed to be found. The notions of what makes optimization problems difficult to solve are established in the theory of computational complexity (Garey and Johnson 1979; Papadimitriou and Steiglitz 1982; Tovey 2002), the approximation theory (Powell et al. 1981) and in the study of non-linear systems (Luenberger, Ye, et al. 2016).

To summarize, in many optimization problems of practical relevance—such as selecting the routes for a fleet of vehicles, distributing a budget across the assets of an investment portfolio, and selecting the parameter values that result in the desired properties for a design—the task of finding the optimal solution can be extremely difficult with today's solution techniques and available computing power. Therefore, in these cases, we have to rely on methods that are capable of finding solutions that are good enough for whoever (the person/organization) is interested in solving the optimization problem, regardless of whether the solutions are optimal or not. Metaheuristics, which are described in the following, are such methods.

---

the problem can be queried but it is not explicitly formulated (Audet and Hare 2017).

2.2 Metaheuristics

Broadly speaking, metaheuristics are optimization techniques that extract information from the search space and use it to direct the search towards areas where high quality solutions can be found. Most metaheuristics have the following characteristics: they are *iterative*—that is, solutions are constructed/perturbed based on starting points or complete initial solutions by an optimization process that consists of a number of steps that repeat for multiple iterations; they use *randomization*—that is, they make use of random variables in one or more of their components; and they have a *user-defined termination criterion*—e.g., reaching a maximum amount of computational time or obtaining a solution of a minimum desired quality.

Metaheuristics can be classified in many different ways. For example, they can be *constructive* or *perturbative*, depending on the way they create new candidate solutions; they can be *memory-based* or *without memory*, depending on whether they memorize solutions (or solution components) or not; and they can also be *metaphor-based* or *non metaphor-based*, depending on their source of inspiration. However, one of the most frequently used ways to tell them apart is given by the number of solutions handled in each iteration, that is, by distinguishing between those that are *single-solution* and those that are *population-based*.

In single-solution metaheuristics, the optimization process is based on one single solution that is iteratively improved by making small changes to it. Examples of this type of metaheuristics are tabu search (Glover 1989), simulated annealing (Černý 1985; Kirkpatrick et al. 1983) and iterated local search (Ramalhinho Lourenço et al. 2002). Differently, population-based metaheuristics maintain multiple solutions in parallel that are combined to create new solutions. Most swarm intelligence (Blum and Merkle 2008; Bonabeau et al. 1999; Dorigo and Birattari 2007; Kennedy et al. 2001) and evolutionary algorithms (Fogel et al. 1966; Holland 1975; Rechenberg 1973; Schwefel 1977, 1981) belong to this class.

Population-based and single-solution metaheuristics have different, often complementary, optimization capabilities. For example, while population-based metaheuristic are typically better at exploring the search space and quickly identifying some of its most promising regions, single-solution metaheuristics tend to be more effective when it comes to improving existing solutions. Thus, it is quite common to combine these two types of metaheuristics in order to

produce metaheuristic implementations that are better equipped to perform robust optimization. Two ways of doing so are *component-based hybrids*, where a population-based metaheuristic includes a single-solution metaheuristic as an additional component in its procedure, and *algorithm-sequential hybrids*, where one metaheuristic is executed after the other with the output of a metaheuristic becoming the input of the next metaheuristic in the sequence (Blum and Roli 2008; Talbi 2013).

Although the vast majority of metaheuristics share the same high-level characteristics (i.e., *iterative*, *randomization* and *user-defined termination criterion*—as mentioned above), they can differ in all kinds of aspects, including the specific mechanisms they use to sample the search space, the way they organize the search process, and the mechanisms they use to control the exploration vs. exploitation trade-off. In a number of cases, the way these aspects are designed in the metaheuristics is by taking inspiration from natural, social or human-made processes. The metaheuristics that have been created using the "inspiration-based" approach are commonly referred to as *metaphor-based* metaheuristics (Sörensen 2015).

Indeed, the approach of looking for inspiration in other fields of knowledge has been very important for designing some of the best performing metaheuristics currently available in the literature. Already in the 1970s, the use of naturally occurring optimization processes—such as evolution by natural selection that inspired *evolutionary computation* (Fogel et al. 1966; Holland 1975)—to formulate new optimization algorithms became appealing to researchers in the areas of computer science and engineering. However, it was in the 1980s and early 1990s that the approach of looking for inspiration in other fields of knowledge began to be explored vigorously in the field of optimization and became a major driver of its development. Notably, during these decades, some thermodynamic principles were used to develop *simulated annealing* (Černý 1985; Kirkpatrick et al. 1983), the foraging behavior of some ant species to develop *ant colony optimization* (Dorigo 1992b; Dorigo et al. 1991b, 1996), and the dynamics and social interactions of bird flocks to develop *particle swarm optimization* (Eberhart and Kennedy 1995; Kennedy et al. 2001; Kennedy and Eberhart 1995).

Because of their success, these metaphor-based metaheuristics are currently among the most extensively studied techniques in the optimization literature, and we have therefore a reasonably good understanding of the way many of them work. In fact, thanks to these studies, we now clearly understand that the reason for their success has little to do with the fact of having found a new source

of inspiration and much with the fact that they introduced novel and useful algorithmic concepts that could be conveniently used for optimization purposes. Unfortunately, this has not been understood by a part of the metaheuristics research community that, as we already mentioned in the introductory chapter, continues to propose "novel" metaphor-based metaheuristics whose sole novelty is in the metaphor and terminology used to describe them.

The next sections present an overview of some selected metaheuristics inspired by natural behaviors. In particular, we focus on *evolutionary computation* (EC) and *swarm intelligence* (SI), since they comprised some of the most successful and well-studied metaphor-based metaheuristics. These metaheuristics are also relevant for the analyses that we present in Chapters 5 and 6. For each of them, we present their source of inspiration, describe their main metaheuristic components, and explain how these components are used to perform optimization.

2.2.1 Evolutionary Computation

Evolutionary computation (EC) (Fogel et al. 1966; Holland 1975; Rechenberg 1973; Schwefel 1977, 1981) is one of the oldest metaphor-based metaheuristic paradigms, whose foundations were inspired by the phenomenon of natural evolution in biological species, such as the Darwinian survival of the fittest and the genetic inheritance. According to natural evolution, species are in a constant search to find useful adaptations to an environment that changes and that is often complicated. Over time, these useful adaptations become traits that the species develop, which are embodied in their chromosomes and represent the "knowledge" that they have gained and allows them to thrive (Michalewicz and Schoenauer 2013). The general procedure of an EC metaheuristic, also known as *evolutionary algorithm* (EA), is the following:

As shown in Algorithm 1, in an evolutionary algorithm, solutions to the optimization problem are represented as a "population" of "individuals" that compete for survival (i.e., to remain in the set of solutions that make up the "population"), where the "fittest" individuals (i.e., those with the best solution quality) are favored to "reproduce" in order to create a new "generation" of solutions (i.e., a new set of solutions that may replace the existing ones and pass to the next iteration of the algorithm). Some of the most well-known EC metaheuristics are *evolution strategies* (Rechenberg 1971; Schwefel 1977), *genetic algorithms* (Goldberg 1989; Holland 1975), *genetic programming* (Koza 1992) and

Algorithm 1 General procedure of an evolutionary algorithm

```
1: begin
2:     initialize population of solutions
3:     evaluate population
4:     while termination condition not met do
5:         select individuals for recombination
6:         apply recombination operator
7:         apply mutation operator
8:         evaluate new individuals (a.k.a. offspring)
9:         select individuals for survival
10:    end while
11:    return best solution found
12: end
```

differential evolution (Storn and Price 1997). In the following, we elaborate on evolution strategies and differential evolution.

Evolution Strategies (ESs) (Rechenberg 1971, 1973; Schwefel 1977, 1981) are among the first evolutionary algorithms proposed mainly to solve continuous optimization problems. ESs use real-valued vectors to represent solutions and, similarly to other EAs, they iteratively apply a number of *evolutionary operators* (or just *operators*) to stochastically sample new solutions. The operators typically used in ESs are:

- *parental selection* – choice of the solutions (*parents*) at the beginning of each iteration that will be used to create new solutions (*offspring*);
- *recombination* – mechanism used to combine the information of two or more *parents* in order to create one or more *offspring*;
- *mutation* – the process of applying a small perturbation to *offspring*;
- *survival selection* – choice of the solutions that will pass to the next iteration (a.k.a. *generation*).

The *parental selection* operator can be implemented using a *deterministic* scheme (one or more specific individuals are selected) or a *stochastic* scheme (individuals are selected according to a probability distribution based on their quality value). The quality value is usually called *fitness* value of an individual. One of the first stochastic parental selection schemes proposed in the literature is *fitness proportional* (Bäck et al. 1997; Holland 1975) and consists in assigning to each individual a probability of being selected that is proportional to its solution quality. Among the deterministic schemes, a typical option is the

so-called *fitness-based* selection, in which only individuals with similar solution quality are matched together to produce offspring (Hansen et al. 2015). Since *fitness-based* selection drives the evolution process towards the best individuals, it is often used when survival selection is *non-elitist* (see below).

The *recombination* operator can be implemented in many different ways. Among the most used recombination operators are *discrete recombination*, where the kth component of the offspring is taken from either of the parents, *intermediate recombination*, where the offspring is the result of computing the arithmetic average of the parents' kth component, and *weighted recombination*, which is similar to *intermediate recombination* but parents can have different weights. Although recombination was widely used in early ESs variants, it is considered optional in most recent ESs implementations.

In ESs, the *mutation* operator is the most important algorithm component and it is commonly implemented by adding a point symmetric perturbation (e.g., random numbers drawn from a multivariate Gaussian/Cauchy/Lévy distribution) to the result of recombination or, if recombination is not used, to the result of parental selection. In practice, most ESs employ the Gaussian distribution to create a perturbed vector $\vec{u}'$, such as the well-known *spherical/isotropic* mutation, which is defined as follows:

$$\vec{u}' = \vec{u} + \mathcal{N}(0, \mathbf{C}), \tag{2.1}$$

where $\vec{u}$ is a vector representing an individual and $\mathcal{N}(0, \mathbf{C})$ is the Gaussian distribution with zero mean and covariance matrix $\mathbf{C} \in \mathbb{R}^{n \times n}$. In the *spherical/isotropic* mutation, $\mathbf{C}$ is proportional to the identity matrix $\mathbf{I}$, and therefore, the Gaussian mutation component is often indicated as $\mathcal{N}(0, \mathbf{I})$.

ESs use the mnemonic notation $(\mu \stackrel{+}{,} \lambda)$ to indicate the way in which *survival selection* will be implemented in the algorithm, where μ and λ are two positive integers that represent, respectively, the number of parents at the beginning of the iteration and the number of offspring generated at each iteration. The symbols "$+$" and "$,$" are used to specify whether survival selection is *elitist* $(\mu + \lambda)$ or *non-elitist* (μ, λ). In the $(\mu + \lambda)$–ES, the next generation is generated by selecting the best μ solutions from the set of $\mu + \lambda$ individuals, whereas in the (μ, λ)–ES the next generation is generated by selecting the best μ solutions from the set of λ offspring.

Arguably the most successful ES is the (μ, λ)–*evolution strategy with covariance matrix adaptation* (Hansen 1997; Hansen and Ostermeier 2001), commonly

know just as CMA–ES, in which the complete covariance matrix of the normal mutation distribution is adapted at execution time. Two variants of CMA–ES that have been proposed to further improve its performance are:

- the *separable*-CMA–ES (Ros and Hansen 2008) — which is a low complexity variant intended for separable continuous functions, where the covariances are assumed to be zeros; therefore, instead of the full covariance matrix, this variant uses a diagonal matrix;
- the *restart*-CMA–ES *with increasing population size* (Auger and Hansen 2005) — which is a variant where the population is doubled after every restart of the algorithm. The algorithm restarts if the range of improvement of the (i) the best solution, (ii) all function values of the most recent generation, or (iii) the standard deviation of the normal distribution do not improve for a number of iterations within a certain threshold.

To this day, CMA–ES remains as one of the best performing metaphor-based metaheuristics available in the literature, and it is commonly employed in experimental comparisons as a benchmark algorithm.

Differential Evolution (DE) (Price et al. 2005; Storn and Price 1997) is a more recent evolutionary approach proposed for the approximate solution of continuous optimization problems that introduced various new concepts to the EC literature. Similarly to ESs, DE starts by sampling the search space with a "population" of randomly created "individuals", that is, real-valued vectors that are solutions to the optimization problem at hand. A minor difference of DE with most EAs is in the order in which the evolutionary operators are applied. Whereas in most EAs the typical order is: parental selection, then recombination, then mutation, and then survival selection (see Algorithm 1); in DE, the mutation operator is applied before recombination.

The main novelty of DE is the way in which the mutation operator is defined, called *differential mutation*, that is similar to the moves in the Nelder-Mead simplex search method (Nelder and Mead 1965). In DE, the mutation operator consists of selecting two existing solutions, computing their vector difference, multiplying their difference by a scaling factor, and adding this to a third vector that is selected among the existing solutions and that is different from the two vectors initially chosen. More formally, in DE, the mutation operator is defined as follows:

$$\vec{m}^{\,i} = \vec{x}^{\,a} + \beta \cdot (\vec{x}^{\,b} - \vec{x}^{\,c}), \tag{2.2}$$

where $i = 1, \ldots, n$ denotes ith individual in a population of n solutions, $\vec{x}^a$, $\vec{x}^b$ and $\vec{x}^c$ are three different vectors chosen from the population, and β is the scaling factor. It is important to note that, while vector $\vec{x}^a$, which is referred to as *base vector*, can be selected in many ways, in most cases, it has to be different from the solution in the population for which it is targeted, that is, vector $\vec{x}^i$ in the current population, that is referred to as *target vector*. The result of applying Equation 2.2 is a vector called *mutant vector*, indicated as $\vec{m}^i$, and it is one of the most important concepts in DE.

The creation of the mutant vector is followed by the application of the recombination operator, in which the mutant vector, $\vec{m}^i$, is recombined with target vector, $\vec{x}^i$, in order to create a new vector, $\vec{u}^i$, that is referred to as *trial vector*. The equation describing how apply the recombination operator and obtain the trial vector is the following:

$$u^{i,k} = \begin{cases} m^{i,k}, & \text{if } (\mathcal{U}[0,1] \geq p_a) \vee (k = k^i_{rand}) \\ x^{i,k}, & \text{otherwise} \end{cases}, \forall k, \forall i, \tag{2.3}$$

where $k = 1, \ldots, d$ allows to iterate between the values of the vectors, $\mathcal{U}[0,1]$ is a random number sampled from a uniform distribution, p_a is a user-selected parameter in the range $[0,1]$ that controls the fraction of values copied from the mutant vector into the trial vector, and k^i_{rand} is a randomly chosen dimension that ensures that the trial vector is not a duplicate of the target vector. The newly generated trial vector $\vec{u}^i$ only replaces the target vector $\vec{x}^i$ in the population if it has better quality, otherwise is discarded. Also, as indicated in Equation 2.3, the mutation and recombination operators are iteratively applied for every solution in the population.

The version of DE that we have described above is known in the literature as the "classic" DE, and it is typically referred to using the mnemonic *DE/rand/1/bin*. After "DE", the second term indicates the way in which the base vector is chosen, the third term indicates the number of vector differences that are added to the base vector, and that fourth term indicates the number of values donated by the mutant vector. Some examples of popular variants of DE indicated using their mnemonics are: *DE/rand/1/bin* (classic DE), *DE/best/1/bin with uniform jitter*, *DE/target-to-best/1/bin*, and *DE/rand/1/either-or*.

In these DE variants, the second term (i.e., the choice of the base vectors) is either "rand", meaning that they are chosen at random from the population, "best", meaning that the best-so-far solution is used as base vector, or "target-

to-best", meaning that base vectors are chosen to lie on the line defined by the target vector and the best-so-far solution. Also, in all cases, the third term is "1", which means that there is only one vector difference being added to the base vector; however, it is not uncommon to add "2" or more vector differences. The fourth term specifies the type of recombination being used, the options showed in the examples are "bin", which means that the number of values donated by the mutant vector follows closely a binomial distribution (as in Equation 2.3), and "either-or", which means that the trial vector is either a three-vector recombination or a randomly chosen population vector to which a randomly chosen vector difference has been added. Finally, the term "uniform jitter" indicates that the scaling factor becomes a random variable β^k that is sampled anew from a normal distribution $\mathcal{N}(0,1)$ for each dimension k of a vector.

2.2.2 Swarm Intelligence

Swarm intelligence studies natural and artificial systems composed of a large number of individuals that coordinate using decentralized control and self-organization to perform tasks or solve problems that are normally too demanding for a single individual (Bonabeau et al. 1999; Dorigo 2001; Dorigo and Birattari 2007). Swarm intelligence has therefore a double nature. On the one hand, it is a scientific discipline that seeks to understand how social animals coordinate their activities, such as ant colonies cultivating plants, fish schools moving in milling or bait ball formations, and bacteria building living structures called biofilms. On the other hand, it is an engineering discipline that tries to design and implement systems that can solve problems of practical relevance and that does so by taking inspiration from the observations and models proposed by the swarm intelligence scientific studies.

The study of swarms of artificial agents that cooperate to solve difficult optimization problems has led to many innovative metaheuristics. Some of the most representative ones are: *ant colony optimization* (Dorigo 1992b; Dorigo et al. 1991b, 1996; Dorigo and Stützle 2004), *particle swarm optimization* (Eberhart and Kennedy 1995; Kennedy and Eberhart 1995), and the *artificial bee colony* algorithm (Karaboga et al. 2005; Karaboga and Basturk 2007). Before discussing these metaheuristics in detail, it is worth noticing that, unlike ESs and DE, where the metaphor of natural evolution is at the basis of the two approaches, in the case of SI, each of the metaheuristics has its own source of inspiration

and specific concepts. In the following subsections, we begin the description of each SI metaheuristic by discussing first its biological inspiration, which allows to understand more clearly the motivation of their authors to consider these behaviors to devise metaheuristic algorithms.

Ant Colony Optimization (ACO) (Dorigo 1992b; Dorigo et al. 1991b, 1996; Dorigo and Stützle 2004), first proposed in 1991, was developed for solving hard combinatorial optimization problems. The first ACO algorithm, called Ant System (AS) (Dorigo 1992b; Dorigo et al. 1991b), was inspired by the work of Deneubourg et al. (Deneubourg et al. 1990; Goss et al. 1989) who studied colonies of Argentine ants and found that they are capable of finding shortest paths between their nests and food sources by depositing pheromones on the ground and choosing their way using a stochastic rule biased by their perceived pheromone intensity. In AS, Dorigo et al. showed that, in a way similar to real ants, artificial ants are capable of building shortest paths on a graph via an iterative construction process in which new nodes are selected and added to partial paths using a stochastic selection mechanism that is biased by "artificial pheromones", a form of distributed information that is iteratively updated by artificial ants and represents the knowledge they have acquired about the quality of the solution components.

To better illustrate how ACO algorithms work, let us consider one iteration of a popular variant of ACO called Ant Colony System (ACS) (Dorigo and Gambardella 1996, 1997b). In the example we present, a population of ants build solutions to a combinatorial optimization problem for which an evaluation function has been defined that can be used to assess their quality.

First, starting from an empty solution, each ant in the population constructs a complete solution to the problem by adding one solution component at a time to a partial solution until it is completed. To do so, each ant k in the swarm has associated a set L_r^k that contains the feasible solution components the ant can add to its partial solution at each construction step r. The mechanism used to select new solution components from L_r^k is given by the transition rule, which differs among ACO variants and is one of the main components in the algorithm. The transition rule of ACS uses a parameter $q_0 \in [0, 1]$ that allows to alternate between two ways of selecting a solution component.

The first selection mode in ACS, which happens with probability q_0, is a simple greedy selection and consists in taking the most "profitable" element in

the set L_r^k. This is done as follows:

$$j_r^k = \arg\max_{h \in L_r^k} \{\tau_h^\alpha \cdot \eta_h^\beta\}, \tag{2.4a}$$

where j_r^k is the solution component to be added to the partial solution constructed by ant k at construction step r, h is a solution component in the set of feasible options L_r^k, τ_h is the pheromone value associated to h, η_h is the value of the heuristic information associated to h, and α and β are two real parameters that control the relative influence of the pheromones and the heuristic information in the equation. The *heuristic information*, η, whose usage is typical in the transition rule of most ACO variants, allows to include problem-specific information into the solution components selection process.

The second way of selecting a solution component in ACS, which happens with probability $1 - q_0$, is a probabilistic one, and consists in assigning probabilities to the solution component in L_r^k and then choosing one based on these probabilities. This selection method is the most widely used across ACO variants and it is defined as follows:

$$p_r^{k,j} = \frac{[\tau_j]^\alpha \cdot [\eta_j]^\beta}{\sum_{h \in L_r^k} [\tau_h]^\alpha \cdot [\eta_h]^\beta} \; \forall j \in L_r^k, \tag{2.4b}$$

where $p_r^{k,j}$ is the probability for ant k of selecting solution component $j \in L_r^k$ at construction step r. Based on the computed probabilities p_r^k, the solution component to be added to the partial solution is chosen using a random proportional mechanism (a.k.a. roulette wheel), so that the elements in L_r^k with higher quality (i.e., those with higher pheromones and heuristic information values) are given higher probability of being chosen. In addition to roulette wheel, there are many other selection mechanisms available in the literature that differ mostly in the type of selective pressure they use—e.g., ranked, tournament, elitist, etc.

In ACS, after a new solution component j has been added to a partial solution, the pheromone value of the component is updated using a local pheromone update procedure as follows:

$$\tau_j = (1 - \varphi) \cdot \tau_j + \varphi \cdot \tau_0, \tag{2.5}$$

where τ_0 is the pheromone lower bound and φ is the local pheromone evap-

oration rate. Both τ_0 and φ are parameters of the algorithm. The goal of Equation 2.5 is to slightly reduce the pheromone value of the newly added component j, so that the probability other ants have of selecting the same sequence of solution components for their solutions is also reduced, thus creating higher diversity in the solutions. In one iteration of the ACS algorithm, Equation 2.4 and Equation 2.5 are applied iteratively by every ant until they have all built a complete solution to the problem.

Last, a global pheromone update procedure takes place in the algorithm, which consists in increasing the pheromone value of the solution components in the best solution. The equation to do so is given by:

$$\tau_j = \begin{cases} (1-\rho)\cdot\tau_j + \rho\cdot\Delta\tau_j^{best} & \text{if } j \in s^{best} \\ \tau_j & \text{otherwise} \end{cases}, \tag{2.6}$$

where $\Delta\tau_j^{best} = F(s^{best}, j)$, $F(\cdot,\cdot)$ is a function that returns the amount of pheromone to be added to each element $j \in s^{best}$, s^{best} is either the iteration-best of the best-so-far solution constructed by an ant, and ρ is a parameter called evaporation rate. The goal of the global pheromone update procedure is to use the best quality solution to provide positive feedback to other ants in the following iterations.

In addition to AS and ACS, many other ant-based algorithms have been proposed in the literature (Dorigo and Stützle 2018; Dorigo and Stützle 2004). However, despite the vast diversity of ideas and ways to implement an algorithm of this kind, a large majority of them can be regarded as particular instances of the ACO metaheuristic framework (Algorithm 2). Some well-known examples of algorithms following the definition of this framework are ACS (Dorigo and Gambardella 1996, 1997b), $\mathcal{MAX}$–$\mathcal{MIN}$ *ant system* (Stützle and Hoos 1996, 2000), AntNet (Di Caro and Dorigo 1998), $\text{ACO}_{\mathbf{R}}$ *for continuous domains* (Socha and Dorigo 2008), $\text{ACO}_{\mathbf{MV}}$ *for mixed-integer variables* (Liao et al. 2014b), and *Pareto*-ACO (Alaya et al. 2007) and *m*-ACO (Iredi et al. 2001) *for multi-objective optimization*.

As shown in Algorithm 2, the ACO metaheuristic framework defines the guidelines to instantiate a constructive, population-based algorithm based on four main algorithmic components: (i) a *stochastic solution construction*, involving the routines needed by the artificial ants to construct solutions, such as the transition rule; (ii) a *local pheromone update*, that is, an optional procedure that

Algorithm 2 ACO metaheuristic

```
1:  begin
2:      set initial parameters
3:      while termination condition not met do
4:          repeat
5:              apply stochastic solution construction
6:              apply local pheromone update                ▷ optional
7:          until construction process is completed
8:          apply daemon actions                            ▷ optional
9:          apply pheromone update
10:     end while
11:     return best solution found
12: end
```

allows local modification of the pheromone trail while constructing solutions; (iii) *daemon actions*, which are optional routines such as local search (Hoos and Stützle 2004) to improve the solutions constructed by the ants; and (iv) a *pheromone update*, involving the routines to modify the pheromone trails in order to ensure an adequate compromise between exploration and exploitation of the search space. In principle, using this framework, ACO algorithms can be applied to any combinatorial optimization problem for which a stochastic solution construction can be defined.

Particle Swarm Optimization (PSO) (Eberhart and Kennedy 1995; Kennedy and Eberhart 1995) was proposed in 1995 for the approximate solution of continuous optimization problems. The first developments of PSO were inspired by studies of the dynamics and social interactions of bird flocks. These studies showed that, by following a few simple rules, simulated bird flocks could display a strong synchronization in initiation of flight, turning while flying, and landing (Heppner and Grenander 1990; Poli et al. 2007). In the proposed PSO algorithm, particles (which correspond to birds in a flock and represent a solution of the considered optimization problem) try to discover the region of the search space where the best quality solutions are located by moving in directions that are estimated based on the best locations that they and their neighboring particles have visited in the past.

In the standard version of PSO (StaPSO) (Shi and Eberhart 1998) (Algorithm 3), the concepts of visual communication, flying coordination, and birds' cognitive capabilities were translated into a computational model composed of three

main elements: (i) a *cognitive component* that allows each particle i in the swarm to memorize the best position it has visited so far, called personal best position $\vec{p}^{\,i}$; (ii) a *social component* that allows a particle to know the best position $\vec{l}^{\,i}$ ever found by any of the particles in its neighborhood; and (iii) a *velocity update rule* and a *position update rule* that specify how the particles move in the search space and that are defined respectively as follows:

$$\vec{v}_{t+1}^{\,i} = \omega\vec{v}_t^{\,i} + \varphi_1 U_{1t}^i(\vec{p}_t^{\,i} - \vec{x}_t^{\,i}) + \varphi_2 U_{2t}^i(\vec{l}_t^{\,i} - \vec{x}_t^{\,i}), \tag{2.7}$$

$$\vec{x}_{t+1}^{\,i} = \vec{x}_t^{\,i} + \vec{v}_{t+1}^{\,i}. \tag{2.8}$$

Algorithm 3 The standard PSO algorithm

```
1:  begin
2:      set initial parameters
3:      t ← 0
4:      repeat
5:          for i ← 1 to size(swarm) do
6:              find l⃗^i in the neighborhood of i
7:              apply velocity update rule          ▷ e.g., Equation 2.7
8:              apply position update rule          ▷ e.g., Equation 2.8
9:          end for
10:         update particles' personal best p⃗
11:         t ← t + 1
12:     until termination criterion is met
13:     return best solution found
14: end
```

As shown in Algorithm 3, the position of the particles (vectors $\vec{x}$), which represent candidate solutions to the optimization problem, are updated in every iteration t of the algorithm by computing a new velocity vector (Equation 2.7) that is added to their current positions (Equation 2.8). The computation of a particle's new velocity makes use of two random diagonal matrices, U_{1t}^i and U_{2t}^i, to introduce diversity to the particle's movement and of three parameters, ω, φ_1 and φ_2, to control, respectively, the influence of the previous velocity (i.e., the particle's inertia), the cognitive component and the social component. The role of vectors $\vec{p}_t$ and $\vec{l}_t$ in the velocity update rule is to combine the knowledge acquired by each particle during the search with the knowledge of the best-informed individual in the neighborhood of the particle.

In PSO, the social component of a particle is determined by its neighborhood, which can be defined in many different ways, allowing in turn to create many

different population topologies. For example, a simple, low-connected topology such as a ring is created by assigning to each particle's neighborhood the two adjacent (i.e., closest) neighbors; while the fully-connected, sometimes referred to as *g*best, topology is created by assigning all particles to the neighborhood of all other particles. In this latter case, the local best particle is also the global best and its position is indicated by $\vec{g}_t$. When using the ring topology, the information about where the best-so-far solution is located spreads slowly among particles, whereas with the fully-connected topology the entire swarm knows immediately the position of the best-so-far solution at each iteration.[2]

Over the years, many improvements and modifications have been proposed for PSO. Some of the best-known variants of this metaheuristic are the *constrained coefficient* PSO (Clerc and Kennedy 2002), the *locally convergent* PSO (Bergh and Engelbrecht 2002), the *locally convergent rotation invariant* PSO (Bonyadi and Michalewicz 2014), the *cooperative* PSO (Bergh and Engelbrecht 2004), the *restart* PSO (García-Nieto and Alba 2011), the *cooperative co-evolution* PSO (Li and Yao 2011), the *group-based* PSO *with random diagonal matrices* (Zyl and Engelbrecht 2016), and the *"charged" particles* (Blackwell and Bentley 2002) and *multi-swarms* PSO (Blackwell and Branke 2004, 2006).

Artificial Bee Colony (ABC) (Karaboga et al. 2005; Karaboga and Basturk 2007), first introduced in 2005, is an optimization technique proposed for tackling continuous optimization problems. ABC was inspired by the self-organization and division of labor observed in honeybee colonies, where individuals specialize in specific tasks and divide accordingly in order to explore different food sources in parallel and exploit more intensively the ones that are more profitable (Karaboga and Akay 2009). In ABC, the food sources exploited by the hive represent candidate solutions to the optimization problem, and the honeybees represent either local search procedures in charge of exploring neighboring solutions, or procedures that are used to create new solutions at random. In plain computational terms, ABC is a combination of parallel local search mechanisms with the occasional introduction of new, randomly created solutions (Aydın et al. 2017b).

As shown in Algorithm 4, the standard implementations of ABC consist of three steps that are repeated iteratively. In the first step, called *employed bees*, each solution $\vec{x}^i$ in the set of candidate solutions (referred to as *reference*

[2]Note that there are many others topologies studied in the literature of PSO, including wheels, lattices, stars, clusters, and randomly assigned edges (Mendes et al. 2004).

Algorithm 4 The standard ABC algorithm

```
set initial parameters
repeat
    apply employed bees step
    apply onlookers bees step
    apply scout bees step
until termination criterion is met
return best solution found
```

solutions in the ABC literature) is used to create a new solution $\vec{x}'^{\,i}$ by modifying a randomly chosen dimension j as follows:

$$x'^{\,i}_k = \begin{cases} x^i_k + \psi^i_k(x^i_k - x^r_k), & \text{if } j = k \\ x^i_k, & \text{otherwise} \end{cases}, \tag{2.9}$$

where k indicates the k^{th} entry of the vector, ψ^i_k is a random number in the range $[-1, 1]$, and x^r is a randomly chosen reference solution different from $\vec{x}^{\,i}$. Also, in the *employed bees* step, when the quality of the newly created solution $\vec{x}'^{\,i}_k$ is higher than the one of $\vec{x}^i_k$, the newly created solution $\vec{x}'^{\,i}_k$ replaces $\vec{x}^i_k$ in the set of reference solutions.

The *onlookers bees* step is similar to the *employed bees* one; the only difference consists in the use of a probabilistic mechanism to select the reference solutions that will be modified. Therefore, as opposed to the *employed bees* step where all the reference solutions in R are used to create new solutions regardless of their quality, in the *onlookers bees* step, the probability of a solution being selected is proportional to its quality, so that the best quality ones are given higher chances.

Finally, in the *scout bees* step, each reference solution has a counter initialized to zero that increases by one when a new solution with lower quality has been created or that re-initializes to zero otherwise. If the counter reaches the value of a parameter λ that limits the maximum number of lower quality solutions that can be consecutively created, the reference solution is removed from the population and a new one is randomly created with a counter initialized to zero.

Even though ABC is a relatively recent approach, it already has a rich literature with many variants and experimental studies investigating ways to enhance its performance. For example, in (Akay and Karaboga 2012), it was showed that the standard version of the algorithm has a slow convergence rate

due to the fact that only one dimension of the solutions is modified at a time. This fact motivated the authors of (Akay and Karaboga 2012) to introduce two changes that have become common design choices in ABC implementations: (i) a parameter MR to probabilistically control the number of dimensions to be modified at each time for each solution, and (ii) the use a parameterized range $[-a, a]$ for computing the value of ψ_k^i, instead of using the fixed range $[-1, 1]$. Additionally, some studies have found a faster convergence speed and an improved performance on specific problems by using alternative initialization schemes, such as chaotic maps (Alatas 2010; Lu et al. 2014; Xiang and An 2013) and opposition-based learning (El-Abd 2011), which produce also a higher diversification of the solutions, thus helping to escape from local stagnation.

There have been as well studies proposing modifications to the main algorithm component of ABC, Equation 2.9, in order to have a better balance between exploration and exploitation of the search space. In general, most changes proposed to this equation are based on ideas employed in other metaheuristic approaches, such as PSO and differential evolution (Storn and Price 1997). For example, the variant proposed in (Xiang and An 2013) replaces vector $\vec{x}^r$ by the best-so-far solution with the goal of enhancing exploitation of the search space, which is similar to the use of vector $\vec{g}_t$ in PSO. Also similar to PSO, in (Gao et al. 2014), the authors proposed to guide the search by a random neighbor or by the global-best solution, whereas in (Li and Yang 2016) the authors gave artificial bees a memory so that they can keep track of their personal best positions. In (Xiang et al. 2014), the authors proposed a search equation for ABC that was inspired by the classical differential evolution (DE) algorithm *DE/rand/1/bin* (Price et al. 2005), in which artificial bees search around the best solution found in the previous iteration.

2.3 Summary

In this chapter, we discussed the fact that many optimization problems of practical relevance are very complex and cannot be tackled in an efficient way with exact methods. Examples of this type of problems are NP-hard problems in the discrete domain, and non-differentiable, multimodal functions in the continuous domain. The computational requirements (time and/or memory) needed by exact methods to solve complex optimization problems to optimality can be exceedingly demanding, making their application either limited to

a small number of cases or altogether impractical. Unlike exact methods, metaheuristics are capable of finding high-quality solutions to these difficult optimization problems in short computational times, but offer no guarantees about the optimality of the solutions.

Another important subject reviewed in this chapter is the use of inspiration from other fields of knowledge to come up with new metaheuristic designs. There are many metaheuristics that have been inspired by natural occurring optimization processes, such as the phenomenon of evolution, which inspired evolutionary computation, and the collective behavior of some species, such as ant colonies and bird flocks, which inspired, respectively, ant colony optimization and particle swarm optimization. In all these cases, the metaphors that inspired the metaheuristics allowed their authors to develop innovative optimization algorithms that introduced novel concepts to the field. However, as we also pointed out in this chapter, in order for this approach to work, there has to be a sound, scientific motivation to use a new metaphor, as well as a careful abstraction of the behavior into algorithm concepts.

Chapter 3

How to Design Metaheuristics: Manual vs. Automatic Approaches

3.1 Manual Design of Metaheuristics

Historically, the creation of new metaheuristics, or of metaheuristics implementations with improved performance, was done by algorithm designers who manually devised new designs or introduced modifications to already existing ones based on their knowledge (either empirical, theoretical or intuitive) and expertise—something typically referred to as *manual design* (López-Ibáñez et al. 2016; Stützle and López-Ibáñez 2019).

Even though manual design has been useful for the implementation of many high-performing metaheuristics, it is becoming less and less efficient over time. Indeed, the design of a high-performing implementation of a metaheuristic requires both to choose among large sets of different possible metaheuristic components and to fine-tune the value of their parameters, and it is often unrealistic to rely on the developers' knowledge and expertise to perform these tasks efficiently. This is because human algorithm designers are biased by their previous experience and limited in the number of designs they can try, making the design process time-consuming and error-prone. In practice, the main disadvantages of manual design are that it limits the *flexibility* with which new metaheuristic designs can be explored and the *generality* of the developed metaheuristics. Additionally, the manual design process is often directed by *subjective* choices that might make the design process difficult to understand a posteriori.

The *flexibility* limitation is caused by the fact that metaheuristics have been

traditionally conceived as monolithic blocks—that is, they have a predefined and often rigid structure. This rigid structure constrains the options for modifying the metaheuristics' behavior: the designer can adjust the metaheuristic's parameter values or redefine the metaheuristic's components (Bezerra et al. 2016). However, the many high-performing hybrid metaheuristics implementations that exist today (Blum and Roli 2008; Maniezzo et al. 2022; Talbi 2002, 2013) have highlighted the importance of being able to propose metaheuristic designs that go beyond the bounds of the original monolithic block. Unfortunately, doing so manually can be very difficult.

The *generality* limitation is caused by the fact that in most cases the initial development of a metaheuristic is done with some application in mind; the performance of the developed metaheuristic tends therefore to be very good for the specific problem or problem class for which the metaheuristic was developed, but it will often be not so good for other problems or problem classes. In general, a metaheuristic's performance tends to drop considerably when it is applied to a problem that has very different characteristics from those for which it was originally developed, or when the optimization scenario has specific constraints that were not considered in the initial metaheuristic design, such as being allowed to run only for a limited time or being unable to use problem-related information (e.g., black-box scenarios or ill-defined real-life problems) (Blackwell and Branke 2006; Hansen et al. 2011). In those cases, a new implementation of the metaheuristic has to be designed for the new problem in order to obtain good results.

Unfortunately, the process of manually adjusting a metaheuristic so as to propose a new metaheuristic implementation is a laborious task riddled with pitfalls. First of all, as we said above, the designer of the new metaheuristic implementation may start by trying to enhance the behavior of the metaheuristic by using different parameter values or by modifying one or more of the metaheuristic's components in the implementation that, according to his/her knowledge and expertise, may be hindering the metaheuristic's performance. If this process, which depends solely on the ability of the human designer, does not produce the desired outcome, then, the only option might be to design and implement a *new* metaheuristic from scratch. This latter task, however, is even more challenging than the previous one, since the algorithm designer now needs to try to come up with a better design guided, once again, only by his/her knowledge and previous experiences.

Finally, manual design makes the design of a metaheuristic a *subjective*

process, where the rationale behind certain design decisions remains hidden in the mind of the human designer and what is learned by the designer is, therefore, not shared with the rest of the research community. In practice, manual design consists in choosing, by trial and error, a good algorithm design from a large set of options. To make the task more manageable, algorithm designers tend to not consider certain types of design that, based on their knowledge, they believe they would not work for the problem at hand. However, which designs are not considered and why they are not considered is rarely written in published technical papers; as a result, most of the times information on the designs that have been tried and that were not particularly successful gets lost. This is unfortunate because having access to this type of "negative" information would be useful to guide the creation of new designs in the future.

3.2 Automatic Design of Metaheuristics

As the need to solve increasingly complex problems more efficiently has grown, so has the need for better and more efficient problem solving methods. This has motivated many researchers in the field to look for alternative design approaches that are not subject to the downsides of manual design. In particular, one of the main goals of the research done in this direction has been to reduce the heavy reliance on human algorithm designers, which makes the design process biased, time-consuming and error-prone. Automatic algorithm design methods are a powerful alternative to manual design. These methods remove the need for human intervention by exploiting recent advances in automatic algorithm configuration methods.

The automatic design of metaheuristics implementations is a relatively new paradigm, in which the creation of a metaheuristic implementation is tackled as an optimization problem that consists in finding a combination of metaheuristic components and parameter settings that will perform very well when applied to the optimization problem considered. To do so, automatic design methods for metaheuristic implementations rely on two main ingredients: a *design space*—that is, the set of all possible metaheuristic' designs that can be obtained by combining metaheuristic components and parameters settings; and an *automatic configuration tool*—that is, a tool that allows the exploration of the design space of the metaheuristic. In recent years, a number of metaheuristic software frameworks have been proposed that greatly facilitate the use of the automatic

design methods to create high-performing metaheuristic implementations.

In the metaheuristics literature, the methods that target the design of metaheuristic implementations as an optimization problem are sometimes referred to as hyper-heuristics. A modern definition of the term hyper-heuristic is the following: "a search method or learning mechanism for selecting or generating heuristics to solve computational search problems" (Burke et al. 2013). Initially, however, the research on hyper-heuristics was not focused on the automatic design of metaheuristic implementations, but rather, on the selection of performing implementation from a portfolio of pre-existing metaheuristic implementations, the so-called "heuristics for choosing heuristics" for combinatorial optimization problems. Nowadays, the automatic design of metaheuristics is approached by hyper-heuristics in the same way as automatic design methods, i.e., by defining a metaheuristic design space and using an optimization algorithm to explore it and find a performing design. In fact, in the vast majority of cases, the only difference between automatic design methods and hyper-heuristics is that hyper-heuristics explore the design space using genetic programming (Koza 1992; Sabar et al. 2013).

3.2.1 Metaheuristic Design Space: The Component-Based View

In order to define a *metaheuristic design space*, the first step is to come up with a component-based view of the considered metaheuristic. To do so, the algorithm designer first identifies the many different ways in which a metaheuristic's components can be implemented (e.g., by studying the different implementations of the metaheuristic that have been proposed in the literature), and then groups them together based on their functionality. The components obtained in this way define the metaheuristic design space and are combined using an automatic configuration tool (see next section). The automatic configuration tool considers these components as parameters—that can be *numerical*, *categorical*, and *subordinate*—to be optimized. Numerical parameters are the classical parameters whose value is either a real or an integer number—e.g., the mutation rate in EC, the evaporation rate in ACO, or the particle's inertia in PSO. Categorical parameters are alternatives for a particular component's functionality—e.g., the recombination operator in EC, the solution construction rule in ACO, or the population topology in PSO. Finally, subordinate parameters are those that are only necessary for particular values of other parameters—e.g., if, in ACO, the $\mathcal{MAX}$–$\mathcal{MIN}$ Ant System pheromone update rule is selected, then the

subordinate parameters controlling the pheromone lower and upper bounds should also be selected. All these parameters form the parameter configuration space **C** that will be used by the configuration tool, as explained in the next section.

3.2.2 Automatic Configuration Tools

Automatic configuration tools (ACTs) were initially developed to automatically select the parameter values in parameterized software in order to increase its performance as much as possible (Audet and Orban 2006; Nannen and Eiben 2006). However, over the years, more general-purpose ACTs have been proposed that allow also to select the algorithm components of the implementation. The use of ACTs took off in the last decade not only because they generate algorithms that, being tailored for a specific problem, most of the times have a very good performance, but also thanks to the higher availability of cheap computing power, as they can be very computationally expensive. The working mechanisms of ACTs are diverse, ranging from experimental design techniques to surrogate-model based approaches. The specific mechanisms implemented in the ACT determine how computationally-intensive it is, the type of parameters it can handle, and the type of post-configuration analyses that can be conducted.

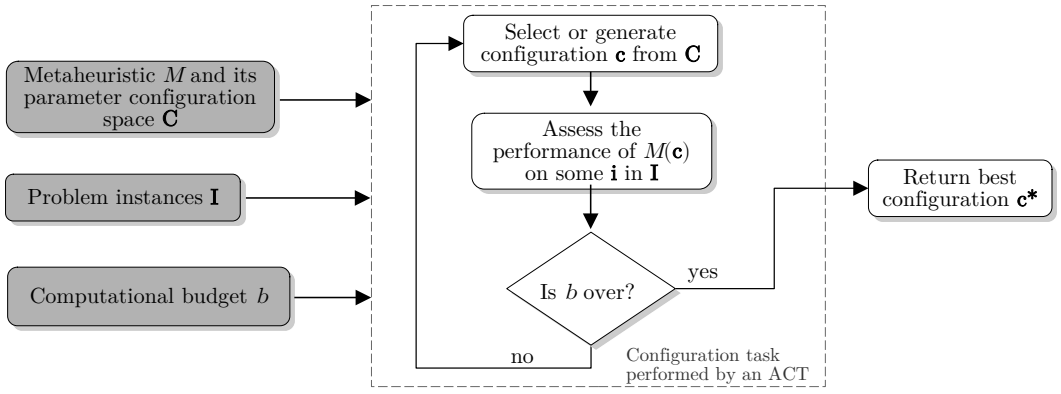


Figure 3.1: General workflow followed by an automatic configuration tool (ACT) used to configure a metaheuristic

The general workflow followed by ACTs is depicted in Figure 3.1. Given a parameter configuration space **C**, an iterative process is carried out in which the metaheuristic M being configured is executed with different parameter

configurations **c** on the set of test instances **I** until a given computational budget **b** is fully used. The different approaches that have been investigated so far to develop ACTs can be categorized as follows:

- *Experimental design techniques*, that are based on the use of statistical techniques to evaluate aspects such as the statistical significance of performance differences; an example of these techniques is CALIBRA (Adenso-Dıéaz and Laguna 2006).

- *Heuristic search techniques*, that consist, as their name suggests, in the application of metaheuristics to tackle configuration tasks. Examples are ParamILS (Hutter et al. 2009), which implements an iterated local search in the parameter configuration space; and the work presented in (Yuan et al. 2012), where the covariance matrix adaptation evolution strategy (CMA-ES) (Hansen and Ostermeier 2001) is used for a configuration task of numerical parameters.

- *Surrogate-model based techniques*, that try to predict the shape of the configuration landscape based on previous execution of the algorithm with the goal of avoiding to waste executions on the unpromising regions. The best-known technique of this type is the sequential model-based algorithm configuration (SMAC) (Hutter et al. 2011).

- *Iterated racing approaches*, that are based on the idea of performing sequential statistical testing using the Friedman test and its related post-tests in order to create a sampling model that can be refined by iteratively "racing" candidate configurations and discarding those that perform poorly. Various racing algorithm are implemented in the `irace` package (López-Ibáñez et al. 2016).

While ACTs differ mostly in the way they approach the automatic configuration problem and in their generality, there are also practical differences that may be important for the user. For example, when used out-of-the-box, iterated racing approaches, such as `irace`, can impose a higher computational overhead during the configuration process than surrogate model-based approaches, such as SMAC, thus making this latter more suitable for configuration scenarios with very expensive objective functions. However, both `irace` and SMAC and other actively maintained ACTs allow nowadays to make changes to their sampling

models in order to reduce computational times in expensive configuration scenarios or to perform a more intensive search if computational time is abundant. Another important difference between ACTs is the use of early termination mechanisms for poorly performing configurations, also called *capping* or *adaptive capping* (Hutter et al. 2009; López-Ibáñez et al. 2016), which help to make a more efficient use of the computational time available, and are particularly useful in those cases where the optimization problems involve time-related objective functions. Finally, a common feature of most ACTs is that they provide lots of useful data that can be used for conducting post-configuration analyses, such as a parameter importance (Hutter et al. 2014; Pérez Cáceres et al. 2017) and ablation analysis (Fawcett and Hoos 2016).

3.3 Summary

In spite of its inherent disadvantages, the use of *manual design* is still by far the most common way to approach the creation of new metaheuristics. In particular, manual design makes it difficult to explore new metaheuristic designs when the number of available design choices is large, which in turn limits the generality with which the metaheuristic can be applied to broader classes of problems. Additionally, in manual design, algorithm designers make numerous choices that are rarely registered for future consideration, which makes the design process subjective and typically quite difficult to reproduce.

Currently, one of the most efficient alternatives to manual design is *automatic design*. In automatic design, the process of designing a metaheuristic implementation is formulated as an optimization problem that consists in finding the combination of components and parameter values that offer the best performance. To apply the automatic design approach, the first step is to define the design space of the metaheuristics using a component-based view. The design space is an abstraction of the many different ways in which a metaheuristic can be implemented, which allows both to replicate known designs as well as to explore new ones never considered before. To automatize the task of exploring a metaheuristic's design space, researchers have developed a number of automatic configuration tools (ACTs). Some state-of-the-art ACTs, such as `irace` and SMAC, have been increasingly used in experimental studies, which show that they are capable of creating high-performing implementations, even in those cases where the design space is large and complex.

Chapter 4

The "Novel" Metaphor-Based Metaheuristics Issue

In addition to the general issues with manual design discussed in Chapter 3, there is the problem that manual design has often been (mis)guided by optimization processes observed in natural systems that were used as a source of inspiration. Even though this approach was very successful in the early days of metaphor-based metaheuristics, when very innovative and well-performing metaheuristics—such as evolutionary computation, ant colony optimization, simulated annealing, and particle swarm optimization—were proposed, this is no longer the case today. Indeed, because of the success of these early metaphor-based metaheuristics, many researchers started to create "novel" metaheuristics using metaphors for which there was no clear mapping between an optimization process observed in the inspiring natural behavior and the optimization process implemented in the corresponding "novel" metaheuristic.

As a result, in the last few decades, hundreds of metaphors from the most diverse set of natural, artificial and even supernatural behaviors, have been used to develop "novel" metaphor-based metaheuristics. In most cases, these metaheuristics have been created based on simplistic mathematical models that vaguely match the behaviors that inspired them and have been presented using overemphasized metaphoric descriptions that make their understanding unnecessarily difficult (some examples are given below). Recently, however, the real novelty of some of the most widespread among these "novel" metaphor-based metaheuristics has been the focus of rigorous analyses, which have provided compelling evidence that, rather than being novel, these metaphor-based metaheuristics are either the same, or at best minor variations, of well-

established metaheuristics, and that their only real novelty is in the use of new metaphors and terminology (Armas et al. 2022; Camacho-Villalón et al. 2019, 2022a, 2023, 2020; Piotrowski et al. 2014; Thymianis and Tzanetos 2022; Tzanetos and Dounias 2021; Weyland 2010).

In particular, three main problems have been identified in papers proposing "novel" metaphor-based metaheuristics (Aranha et al. 2022). First, the introduction of useless metaphors that do not have any scientific basis—e.g., *zombies*, *reincarnation*, and *intelligent water drops*—and of new, unnecessary terminology that makes it difficult to understand the ideas proposed in these metaheuristics. Second, the lack of any significant novelty; indeed, it is typically the case that the ideas proposed in these metaheuristics are already known in the field. Third, the use of poor experimental validation and comparison practices, such as comparing "novel" metaheuristics run on recent computers against old algorithms run on old computers, and the use of benchmark testbeds that contain biases that can be exploited by the "novel" metaheuristics, thus favoring their performance.[1]

The negative consequences of the "novel" metaphor-based metaheuristics trend extend well beyond the existence of a few algorithms inspired by far-fetched behaviors that were presented in papers with methodological issues. In fact, the hundreds of "novel" metaphor-based metaheuristics already published in the literature (Campelo and Aranha 2021a) are the result of a pernicious trend grounded on unscientific practices. Unfortunately, this unhealthy trend is still far from being over; this is due, in large part, to the prevalence of manual design as the main way to create metaheuristics. In the remainder of this subsection, we explain in detail why "novel" metaphor-based metaheuristics are so problematic, what are some of their negative consequences, and what efforts have been done so far to address them.

4.1 The Metaphor Rush

Taking inspiration from natural processes has played a major role in the development of many innovative solutions. In particular, in science and technology, there are many instances of the use of this approach, a few well-known exam-

[1] An example of this was provided in (Kudela 2022), where it was experimentally demonstrated that the "novel" *slime mold*, *butterfly* and *Harris hawks* metaheuristics, among others, make use of center-bias operators that increase their efficiency when the testbed includes problems that have the optimal solution located in the center of the search space.

ples are synthetic, self-cleaning materials inspired by some plants leaves (Wang et al. 2020), the Japanese bullet trains whose aerodynamic front shape was inspired by a bird's beak (Crandell et al. 2019), and a number of learning and intelligent algorithms, such as those found in the evolutionary computation and swarm intelligence research fields, whose sources of inspiration were discussed in detail in Chapter 2. Additionally, in virtually all domains and fields, it is common to use metaphors to present abstract ideas in order to help convey more easily their meaning, such as the metaphor of *trees* in graph theory that helps to visualize a specific type of data structure, or the fact that we use the word *mouse* to refer to a device that looks like a small rodent attached to our computer (although wireless and ergonomic mice have rendered the metaphor much less effective).

Starting in the mid 2000's, the field of optimization has witnessed a true *rush* to find "interesting" behaviors, mostly from natural and social phenomena, that can be used to devise reputedly "novel" metaheuristics. In the numerous papers proposing this kind of metaheuristics, the authors present their metaheuristics using the following sequence of steps: (i) they start by claiming to have found a new behavior that has applications in optimization; (ii) they present why, in their opinion, the behavior is interesting, followed by an extensive list of other "novel" metaheuristics based on "interesting" behaviors; (iii) they describe their proposed metaheuristic using the terminology of the behavior instead of the one that is normally used in optimization; and (iv) they compare the "novel" metaheuristic with other optimization techniques that are often old and whose performance is much worse than the state of the art.

To give a concrete example, we can consider the *grey wolf optimizer* (Mirjalili et al. 2014). According to its authors, this metaheuristic is inspired by "the way grey wolves organize for hunting" following a "strict social hierarchy". To describe the metaheuristic, the authors introduce a new terminology in which candidate solutions are referred to as "wolves", the three best solutions in the "pack" (i.e., the set of candidate solutions) are referred to as the "alpha", "beta" and "delta" wolves, and the optimum of the problem is the "prey" the wolves are hunting. As shown in (Camacho-Villalón et al. 2023, 2020), the *grey wolf optimizer* metaheuristic is based on the idea of computing, at each iteration, the centroid of a hypertriangle whose vertices are the position of the three best solutions (i.e., "alpha", "beta" and "delta"), and of using the computed centroid to bias the movement of the rest of the solutions (i.e., the "pack"). The performance of the *grey wolf optimizer* metaheuristic was evaluated on a

set of 29 continuous functions, all with low dimensionality and/or with the optimum at the center of the search space, as well as on some classic engineering design problems; it was compared against PSO, DE, and the covariance matrix adaptation evolution strategies (CMA-ES) (Hansen and Ostermeier 2001), with the outcome that it was found to have a similar performance. This similarity in performance, however, is not a surprise, since the "novel" *grey wolf optimizer* metaheuristic was later shown to be nothing more than a simple variant of PSO (Camacho-Villalón et al. 2023, 2020).

Another example of a popular "novel" metaphor-based metaheuristic is *cuckoo search* (Yang and Deb 2009), which is described using the metaphor of "cuckoo's parasitic behavior". In this metaheuristic, initial solutions are referred to as "cuckoos" while solutions that have been perturbed are referred to as "eggs". The *cuckoo search* metaheuristic is based on the idea that some "cuckoos" lay a number of "eggs" in the "nests" of other birds (which are random points in the search space) and some of these "eggs" will still exist in the next iteration and some other will not. The authors of *cuckoo search* compared their metaheuristic against a genetic algorithm and the first version of PSO. The comparison was run using 13 continuous functions that were all with optimum at the center of the search space and for which the authors did not provide any information about their dimensionality. Because of this, it is difficult to evaluate the quality of the results they obtained. However, this might be considered irrelevant given that, perhaps unsurprisingly for those familiar with evolutionary computation, *cuckoo search* turned out to be an evolution strategy that uses the crossover of differential evolution (Camacho-Villalón et al. 2022a), and therefore does not give any contribution to the state of the art.

4.2 The Consequences of the Metaphor Rush

Over time, the number of papers proposing "novel" metaphor-based metaheuristics has grown to several hundreds—at the time of writing this article, it is estimated there are around 500 metaphor-based metaheuristics (Ma et al. 2023; Tzanetos and Dounias 2021). Very often, the authors of these papers make very strong, and wrong, statements that go from saying that they are proposing a novel technique inspired by some sort of "intelligent" behavior, to claiming they

have found a new and unexplored field.[2] Unfortunately, despite the complete lack of evidence to support such statements and the increasing awareness about the problems that these "novel" metaheuristics are causing in the research field, a part of the metaheuristics community is under the delusion that this is a good approach and continues to actively propose more metaheuristics of this kind.

One of the main problems caused by the constant publication of papers proposing "novel" metaheuristics has been the fragmentation of the literature into dozens of barely distinguishable niches (Aranha et al. 2022; Campelo and Aranha 2021b). A direct consequence of this fragmentation is a confusing literature, in which the same ideas and concepts are being reintroduced over and over again using many different terminologies derived from the use of new metaphors. This, in turn, has made the comparisons of metaheuristics increasingly challenging. Indeed, just to make an example, it is very difficult to compare the optimization capabilities of "grey wolves hunting" with those of "cuckoos laying eggs". But, even worse, when one analyses the mathematical models proposed for these metaheuristics more closely, they turn out to be either the same or minor variations of those proposed in optimization techniques published many years before.

The hundreds of "novel" metaphor-based metaheuristics already published have popularized the wrong idea that having found a new "interesting" behavior and developing a simplistic mathematical model based on it justifies its inclusion in the metaheuristics literature, regardless of whether or not it brings novel and useful ideas to the field. It is very worrying indeed that articles that have serious methodological flaws and total lack of scientific rationale[3] are so constantly able to pass the peer review process of scientific venues, many of which seem—most unfortunately—to be a lot more interested in future citation counts than in publishing quality papers with meaningful contributions.

Finally, there is the damage caused to the reputation and external perception of the field of metaheuristics.[4] In particular, it is causing damage to the fields of swarm intelligence and evolutionary computation that are often used as

[2]See for example: Satish Gajawada. POSTDOC: The Human Optimization. *Computer Science & Information Technology* (CS & IT), CSCP 3 (2013): 183–187. Pdf available at https://www.airccj.org/CSCP/vol3/csit3918.pdf

[3]Consider, for example, the following excerpt from the *intelligent water drops* paper: "*In nature, we often see water drops moving in rivers, lakes and seas... We also know that the water drops have no visible eyes to be able to find the destination (lake or river).*" More on this in Chapter 5.

[4]This issue was brilliantly exposed by Sörensen in (Sörensen 2015) as well as in a parodic (yet insightful) paper titled "A Spectral Approach to Ghost Detection" (https://sigbovik.org/2013/proceedings.pdf), whose reading we strongly recommend.

examples to justify the use of metaphors. The misconception is that, even though, in particular cases and if done properly, a metaphor can guide the design of a successful metaheuristic, this happens very rarely and in most cases it only ends up in unnecessary confusion.

4.3 The Role of the Academic Research System

It would be very hard to explain the existence of a trend such as the one of the "novel" metaphor-based metaheuristics without considering some of the problems that exist in the current academic research system. The academic research system is the framework used by academic researchers to carry out, review and disseminate scientific contributions, and involves the interaction of many different people with many different interests and goals. Some of the problems that exist in the current academic research system are:

- Misuse of incentives: One of the most pervasive problems in the academic research system is the use of incentives (e.g., financial support, professional recognition, academic promotion, etc.) as levers to increase "productivity", which is measured as a function of the number of articles a researcher has published in a certain period of time (Edwards and Roy 2017). This way of measuring productivity, that favors quantity over quality, puts an excessive pressure to publish, creating a phenomenon otherwise known as *publish-or-perish* (Harzing 2010).
- Journal impact factor manipulation: The journals impact factor (JIF) is widely used as a measure of the quality and influence of a journal within the field and, by extension, of the quality of the work produced by the researchers that publish in that journal. Artificially inflating JIFs (either by authors excessively citing some specific sources or by editors asking authors to cite the articles published by their journals) not only results in inaccurate information, but also puts editors under pressure to game this metric in order to "attract" submissions (Ioannidis and Thombs 2019).
- Weakness of the peer-review process: One critical problem of the academic research system is that the peer-review process that we use to control the quality of the publications is not robust enough to detect and prevent dishonest behaviors (Edwards and Roy 2017; Weyland 2010). In order for the peer-review process to work well, it is unrealistically assumed that everyone involved will always play by the rules. Unfortunately, there is an

increasing number of examples showing that this is not always the case (Sörensen 2015; Sörensen et al. 2019).

- Conflicts of interest: The experimental sciences have been shown to be particularly vulnerable to the existence of conflicting interests in the academic research system. While having different interests is not by itself a problem, when the goals of the different parties involved in scientific research are unaligned or conflicting, the consequences can be detrimental, creating problems, such as biases, inaccuracy in the results, and false findings (Ioannidis 2005). The existence of conflicting interests in experimental sciences has been recognized as one of the main underlying causes of the *reproducibility crisis* (Baker 2016; López-Ibáñez et al. 2021).

Even though the problems of the academic research system are widespread and affect different scientific fields in different ways, it is not difficult to see the role they have played in letting the trend of the "novel" metaphor-based metaheuristics grow larger and larger over the years. Indeed, the existence of these problems can help us to understand (i) why some authors can be motivated to propose "novel" metaheuristics in vast quantities, (ii) why most of the papers of this kind include a long list of references to other similar "novel" metaheuristics, (iii) why the papers proposing "novel" metaheuristics are so constantly able to pass the peer-review process of some publication venues, and (iv) why some publication venues may welcome articles proposing "novel" metaheuristics regardless of their real contribution and quality.

4.4 Efforts to Mitigate the Metaphor Rush

So far, most efforts aimed at dealing with the metaphor-based metaheuristics involve raising awareness about the issue and how they negatively affect the metaheuristics research field. One of the earliest efforts in this sense was the publication by Dennis Weyland of the paper "A Rigorous Analysis of the Harmony Search Algorithm" (Weyland 2010), where it was shown, by means of a component-by-component comparison, that *harmony search* is in fact an evolutionary algorithm. The paper by Weyland also identified a number of problems in the actual research system that contributed to let a "novel" metaheuristic like *harmony search* become popular despite its complete lack of novelty. Another notable example is the paper "Metaheuristics—The Metaphor Exposed" by Kenneth Sörensen (Sörensen 2015), which was the first paper

clearly attempting to bring attention to the "metaphor problem" in the field of metaheuristics.

Although these papers barely resonated outside the community that was already aware of the problem, they were a stimulus for other researchers to act and propose possible solutions. Most of these efforts can be categorized as follows: (i) critical analysis of metaphor-based metaheuristics, (ii) modeling frameworks, taxonomies, and metaphor-free descriptions, and (iii) editorial policies.

In category (i) we find the efforts to clarify whether there is any real novelty in these "novel" metaheuristics and obtain insights on the motivation the authors had to use a particular metaphor. Component-based analyses, similar to that of Weyland (Weyland 2010), have been conducted for the following "novel" metaheuristics: *biogeography-based optimization* (Simon et al. 2011), *black hole optimization* (Piotrowski et al. 2014), *intelligent water drops* (Camacho-Villalón et al. 2018, 2019), *grey wolf optimizer*, *moth-flame optimization algorithm*, *whale optimization*, *firefly algorithm*, *bat algorithm*, *antlion optimizer* (Camacho-Villalón et al. 2023, 2020), and *cuckoo search* (Camacho-Villalón et al. 2022a). The conclusion of all these analyses has been clear: there is no novelty in any of these "novel" metaheuristics, since they only have negligible differences with well-established metaheuristics. In a critical paper with a slightly different focus, Melvin et al. (Melvin et al. 2012) showed that the *gravitational search algorithm* is based on a mathematical model that is inconsistent with Newtonian gravity, thus rendering the use of the metaphor of gravity useless. The *gravitational search algorithm* is an example of why using new metaphors without having a sound motivation to do so may actually result in ineffective metaheuristics.

Category (ii) includes the efforts that many researches have done to try and put order in the literature by proposing taxonomies and modeling frameworks that allow to establish similarities among metaheuristics based on patterns in their design (Armas et al. 2022; Cruz-Duarte et al. 2020; Fong et al. 2016; Molina et al. 2020; Stegherr et al. 2020; Thymianis and Tzanetos 2022; Tzanetos and Dounias 2021). The ultimate goal of these efforts is to provide the metaheuristics community with a comprehensive tool that can help to readily identify the components that make up a metaheuristic, and consequently, to have a systematic way to evaluate whether a "novel" metaheuristic is actually novel or not. The main challenge is to incorporate a sufficiently large number of components in the tool so that it is representative of the vast diversity of the metaheuristics and of their implementations; in this way, it can be used to identify virtually

any possible metaheuristic design. While the first steps in this direction have been taken, this is an huge endeavor that will require a great deal of additional research in order to be accomplished.

In category (ii), we also find those papers aimed at better understanding the way metaphor-based metaheuristics work and their relationship with other metaheuristics. Examples of these efforts are the papers by Lones (2014, 2020), in which the author provides an accessible description of some of the "novel" metaphor-based metaheuristics using a metaphor-free terminology. In addition to this, there is an increasing number of papers aimed at quantifying the size of the problem by compiling comprehensive lists of metaphor-based metaheuristics and/or analyzing their performance (Campelo and Aranha 2021a; Kudela 2022, 2023; Ma et al. 2023; Tzanetos et al. 2020).

Finally, category (iii) contains the very few efforts coming directly from editorial policies that explicitly forbid the submission of papers proposing metaphor-based metaheuristics unless the authors can provide compelling evidence that the use of the metaphor contributes to the advancement of the state of the art. Some of the journals that have established this type of policies are *4OR*,[5] *Journal of Heuristics*,[6] *Swarm Intelligence* (Dorigo 2016), *ACM Transactions on Evolutionary Learning and Optimization*,[7] and *Engineering Applications of Artificial Intelligence*.[8] Establishing editorial policies is undoubtedly one of the most effective mechanisms to stop the publication of metaphor-based metaheuristics; unfortunately, in the scientific publication system this approach remains the exception rather than the rule.

4.5 Summary

In the chapter, we discussed the problematic trend of the "novel" metaheuristics based on all kinds of metaphors, which exist in part due to the prevalence of manual design as the main way to create metaheuristics. In order to illustrate why these kinds of metaheuristics are so problematic, we presented two concrete examples of highly-cited "novel" metaheuristics that turned out to lack any novelty, and being therefore only a source of confusion and reiteration of known

[5] https://www.springer.com/journal/10732/updates/17199246

[6] https://www.springer.com/journal/10732/updates/17199246

[7] https://dl.acm.org/journal/telo/author-guidelines

[8] https://www.sciencedirect.com/journal/engineering-applications-of-artificial-intelligence

ideas. Unfortunately, these are just two examples in a long list of other "novel" metaheuristics with similar problems. In Chapters 5 and 6, we present the complete analysis of the two "novel" metaheuristics mentioned in this chapter, as well as the analysis of several others in order to provide further evidence for the thesis that these type of metaheuristics are not truly novel.

While many important efforts have been conducted to address the problem of the "novel" metaphor-based metaheuristic, none of them has been comprehensive enough to solve it definitively. Nonetheless, the number of efforts in this direction keeps growing and many researchers/practitioners are now aware of their negative consequences. Unfortunately, the metaheuristics community has become divided between two conflicting approaches to continue pushing the field forward. On the one hand, even though the use of automatic design methods to create new metaheuristics designs has increased over time and its benefits have already been shown experimentally in many papers, the use of this approach still remains scarce in the literature. On the other hand, the idea of introducing new metaphors is still being used intensively, as evidenced by the alarming regularity with which new papers proposing "novel" metaheuristics based on absurd behaviors are published in the literature.

Part II

Analyses of "Novel" Metaphor-Based Metaheuristics

Chapter 5

"Novel" Metaheuristics for Discrete Optimization

In this chapter, we study the *intelligent water drops* (IWD) metaheuristic and its relation to *ant colony optimization* (ACO). We start by describing the source of inspiration and basic concepts introduced in the two metaheuristics—ACO, in Section 5.1 and IWD, in Section 5.2. Then, in Section 5.3, we perform a component-by-component comparison between ACO and IWD, and show that IWD is, in fact, a particular case of ACO. We also discuss the fact that the metaphor of "intelligent water drops removing soil from the ground of a river" that inspired the IWD metaheuristic does not correspond to any scientific observations about the way erosion works in river systems. Finally, in Section 5.4, we review published research on IWD, and provide compelling evidence that most of the ideas that have been proposed to enhance the performance of IWD were already proposed in the ACO literature.

5.1 Overview of Ant Colony Optimization

Ant colony optimization[1] (ACO) (Dorigo 1992b; Dorigo et al. 1991a,b) is a metaheuristic proposed in the early 1990s that was inspired by the seminal work by Deneubourg et al. (1990) on the *Argentine* ant foraging behavior. Denebourg et al. showed that *Argentine* ants can find a shortest path between their nest and a food source by depositing pheromones on the ground and by choosing their way using a stochastic rule biased by their perceived pheromone intensity.

[1]This section presents a short overview of ACO that is intended for its comparison with IWD. For a more comprehensive description of ACO, see Chapter 2.

Based on Deneubourg's findings, Dorigo et al. showed that, in analogous way to real ants, artificial ants that

- move on a graph representation of a discrete optimization problem, where edges are solution components and where a path on the graph corresponds to a problem solution,
- deposit virtual pheromones on the graph edges (or equivalently on solution components), and
- use pheromones to bias the construction of random paths on the graph,

can find high quality solutions by letting their stochastic solution construction routine be biased by the value of virtual pheromones. Ant System (Colorni et al. 1992; Dorigo et al. 1991b), the first algorithm proposed based on the metaphor of ants foraging behavior was a combination of three main concepts: (i) many interactive agents, also called *artificial ants*, (ii) a reinforcement mechanism to give a *positive feedback* to selected solution component,[2] and (iii) a constructive greedy heuristic to build paths on a graph.

The publication of the seminal algorithm (Dorigo 1992b; Dorigo et al. 1991a,b, 1996) was followed by many variants and improvements (Alaya et al. 2007; Birattari et al. 2006; Blum 2005; Blum and Dorigo 2004; Bullnheimer et al. 1999b; Cordón et al. 2000; Dorigo and Gambardella 1997a; Dorigo et al. 1996; Gambardella and Dorigo 1995; Guntsch and Middendorf 2002; Maniezzo 1999; Socha and Dorigo 2008; Stützle and Hoos 1997); most of these works are summarized in the *Ant Colony Optimization* book (Dorigo and Stützle 2004). Throughout all this literature, ACO has been described as a constructive, population-based metaheuristic composed of three main components: (i) *stochastic solution construction*, involving the routines needed to construct solutions; (ii) *daemon actions*, containing optional routines to improve the solutions constructed by the ants; and (iii) *pheromone update*, involving the routines to modify the pheromone trails in order to ensure the exploration and exploitation of the search space. The main idea is that artificial ants are probabilistic procedures that construct solutions iteratively (i.e., by adding one solution component at a time to a partial solution) based on virtual pheromones and heuristic information.

One iteration of ACO can be described as follows. Starting from an empty solution, an artificial ant implements the stochastic solution construction rou-

[2] As explained in (Dorigo et al. 1991b), *positive feedback* allows ants to generate a process that reinforces itself, that is, the more ants are following a trail, the more attractive that trail becomes for being followed.

tines needed to add solution components until the solution is completed.[3] After the construction phase is over, daemon actions may take place. Daemon actions are routines that cannot be performed by a single ant. They may consist, for example, of a local search procedure that improves the solution constructed by an ant, or of a procedure that deposits an additional amount of pheromone on solution components that belong to solutions with some desirable characteristics. Finally, pheromone update consists in the modification of the pheromones with the goal of biasing the construction process in the following iterations towards better solutions. The pheromone update procedure involves depositing pheromone on the components belonging to good quality solutions, and evaporating pheromone in components producing solutions of lower quality.[4] Several iterations are executed until a termination condition is verified and the algorithm stops. This process is shown in Algorithm 5.

Algorithm 5 ACO metaheuristic with its main components

```
1:  set initial parameters
2:  while termination condition not met do
3:      repeat
4:          apply stochastic solution construction
5:          apply local pheromone update                ▷ Optional
6:      until construction process is completed
7:      apply daemon actions                            ▷ Optional
8:      apply pheromone update
9:  end while
10: return best solution
```

Virtual pheromones—pheromones for short in the following—and heuristic information are the main sources of information used by artificial ants to construct solutions stochastically. Pheromones, indicated by τ, are numerical values associated to solution components that are iteratively modified by ants in order to mark solution components that produce good solutions. The amount of pheromones in the solution components can increase using *pheromone deposit*, or decrease using *pheromone evaporation*. While pheromones represents the knowledge acquired during the algorithm's execution, the *heuristic information*,

[3] The rule for selecting solution components, called *transition rule*, implemented during the stochastic solution construction varies among ACO variants.

[4] In some ACO implementations, the *pheromone update* can be interleaved with the solution construction (e.g., see (Dorigo and Gambardella 1997a; Gambardella and Dorigo 1995)), an example being the *offline pheromone update* implemented in ACS (Dorigo and Gambardella 1997a).

indicated by η and also associated to solution components, is a way to include problem-specific information to guide the search. The use of heuristic information greedily bias the selection of components that have a lower cost in the solution under construction. There are different strategies to weight the relative importance of parameters τ and η; we discuss some of them in Section 5.3.1. Finally, in Table 5.1 that appears below, we list the most important ACO variant, which differ mainly in the way in which stochastic solution construction and pheromone update are implemented.

5.2 Overview of the Intelligent Water Drops Metaheuristic

The intelligent water drops (IWD) metaheuristic was proposed by Shah-Hosseini (2007) as a new problem solving algorithm for combinatorial optimization. The author of IWD says that this metaheuristic is based on the observation of rivers in nature and is explained using a metaphor in which water streams are seen as groups of individual particles (water drops) removing soil from the ground of the riverbed. According to the metaphor of the "intelligent water drops removing soil from the ground of a river", in their journey from a source to a destination, water drops prefer paths with less soil; also, on paths with less soil they move faster, and the faster they move the more soil they remove. Following this self-reinforced mechanism, the water drops are capable of finding shortest paths from a source to a destination. In the words of the author:

> *"In nature, we often see water drops moving in rivers, lakes and seas. As water drops move, they change their environment in which they are flowing ... We also know that the water drops have no visible eyes to be able to find the destination (lake or river). If we put ourselves in place of a water drop of the river, we feel that some force pulls us toward itself (gravity)."*
> (Shah-Hosseini 2007, pp. 3326)

> *"In the water drops of a river, the gravitational force of the earth provides the tendency for flowing toward the destination. If there were no obstacles or barriers, the water drops would follow a straight path toward the destination, which is the shortest path from the source to the destination. However, due to different kinds of obstacles in their way to the destination, which constrain the path construction, the real path has to be different from the ideal path and lots of twists and turns in*

> *the river path is observed." ... "It is assumed that each water drop flowing in a river can carry an amount of soil... The amount of soil of the water drop increases while the soil of the riverbed decreases. In fact, some amount of soil of the river bed is removed by the water drop and is added to the soil of the water drop."*
> (Shah-Hosseini 2008, pp. 195)

> *"A water drop has also a velocity and this velocity plays an important role in the removing of soil from the bed of the rivers ... The faster water drops are assumed to collect more soil than others."*
> (Shah-Hosseini 2008, pp. 196)

Shah-Hosseini (2007, 2008, 2009) translated these ideas into a metaheuristic where water drops: (i) move in discrete steps on a graph representation of the considered optimization problem, where edges are solution components, and each solution component j has an associated amount $soil_j$ of soil; (ii) modify the amount of soil on the solution components (graph edges) as a function of their velocity and of problem specific information called *heuristic undesirability*; and (iii) use the amount of soil associated to solution components to bias the construction of random paths.

The IWD metaheuristic, as described in (Shah-Hosseini 2007, 2008, 2009), is a constructive, population-based optimization technique composed of three components: (i) *stochastic solution construction*, (ii) *local soil update*, and (iii) *global soil update*. In IWD, the water drops have two variables associated: a *velocity* and the total amount of *soil collected*. Moreover, the water drops cooperate to construct solutions incrementally using a probabilistic rule, called *random selection rule*, which is biased by the amount of soil associated to the solution components.

In one iteration of the IWD metaheuristic, these three components are applied as follows. First, during stochastic solution construction, each water drop starts from an empty solution and adds one solution component at a time until the solution is completed. Interleaved with the stochastic solution construction, the local soil update involves two actions after a solution component has been added to a partial solution: (i) a decrease of the soil in the solution component just added, and (ii) an increase of the soil collected in the water drop. In fact, every time a water drop adds a new solution component to the solution it is constructing, it updates its velocity and its total amount of soil collected. Finally, the global soil update procedure updates the soil in solution components of the iteration-best water drop (i.e., the water drop that built the best solution in the

current iteration). The algorithm stops once a termination criterion is met. A high level description of the metaheuristic is given in Algorithm 6.

Algorithm 6 The intelligent water drops metaheuristic

```
1: set initial parameters
2: while termination condition not met do
3:     repeat
4:         apply stochastic solution construction
5:         apply local soil update
6:     until construction process is completed
7:     apply global soil update
8: end while
9: return best solution
```

It is easy to see that, in IWD, the soil associated to the solution components plays the same role as the virtual pheromone in ACO: it biases the stochastic choice of solution components during the stochastic solution construction process. However, differently from artificial ants in ACO, each water drop k has two associated variables: vel^k, that is the velocity of the water drop and represents the quality of the partial solution that it has built so far; and $collected\_soil^k$, that is the soil collected by the water drop while building a solution.

At the beginning of each iteration, the initial velocity, vel^k, of the water drops is always the same, but this variable is updated for each water drop as a function of the soil in the components added to its partial solution. Therefore, it is typically the case that water drops have different velocities at the end of each iteration. Also, the velocity of a water drop is used to compute the amount of soil it collects when adding a new solution component.[5]

The variable $collected\_soil^k$ is used by the water drop to keep a record of the soil collected from the solution components added to the solution that it is constructing. The amount of soil added to $collected\_soil^k$ is also proportional to a value called *heuristic undesirability*[6] divided by the water drop velocity. Finally,

[5]Note that, even though this is counter-intuitive, the amount of soil that a water drop collects when adding a new solution component to the partial solution is different from the amount of soil that is removed from the soil variable associated to the added component. For a detailed example see Appendix A.

[6]The author calls *heuristic undesirability* to the inverse of the *heuristic information* used in ACO. For example, in the traveling salesman problem, ACO's *heuristic information* is commonly defined as $\eta_{ij} = 1/d_{ij}$, where d_{ij} indicates the distance between city i and city j. In IWD, the *heuristic undesirability* is, for the same problem, defined as $HUD_{ij} = d_{ij}$.

as mentioned above, the best water drop updates the solution components at the end of each iteration using the amount of soil collected in its associated variable $collected\_soil^{best}$.

5.3 Comparison Between ACO and IWD

In the previous sections, we have described ACO and IWD starting with their sources of inspiration and describing how these sources of inspiration were abstracted as metaheuristic concepts that can be used to solve optimization problems. In summary, the two metaheuristics consist of the following three main components:

- *stochastic solution construction*: to construct solutions biased by a quantity (pheromone/soil) associated to solution components,
- *local update*: to improve the search by interleaving the construction mechanism with an update of pheromone/soil on the last added solution component, and
- *global update*: to provide a positive feedback via modifications of the pheromone/soil associated to specific solution components.

In this section, we carry out a detailed comparison of the two optimization techniques in order to clarify whether IWD is in fact a new metaheuristic, and deserves therefore to be called a novel approach, or should rather be considered a variant of ACO. To do so, in Table 5.1 we schematically present the metaheuristic components proposed in some of the best-known ACO variants: *ant system* (AS) (Dorigo et al. 1991a,b, 1996), *ant system with Q-learning* (Ant-Q) (Gambardella and Dorigo 1995), $\mathcal{MAX}$–$\mathcal{MIN}$ *ant system* ($\mathcal{MM}$AS) (Stützle and Hoos 2000), *ant colony system* (ACS) (Dorigo and Gambardella 1997a), *approximate nondeterministic tree-search* (ANTS) (Maniezzo 1999); and in the *intelligent water drops* (IWD) (Shah-Hosseini 2009).

However, before presenting the component-by-component comparison of ACO and IWD, we briefly discuss the notions of *soil* and of water drop's *velocity*. This discussion will help the reader understand the analysis presented in this section. In section 5.2, we noted that the role played by the soil value associated to solution components in IWD is very similar to the one played by pheromones in ACO. However, in high-quality solution components, the value of pheromones tend to increase over time, whereas the value of soil tend to

Table 5.1: Main components used in Ant System (AS), Ant System with Q-learning (Ant-Q), $\mathcal{MAX}$–$\mathcal{MIN}$ Ant System ($\mathcal{MM}$AS), Approximate Nondeterministic Tree-Search (ANTS), Ant Colony System (ACS) and the Intelligent Water Drops (IWD). We do not show here the *daemon actions* component commonly integrated in ACO implementations. In the components presented here under, s^k indicates the solution constructed by ant/water drop k, s^{best} indicates the best solution constructed either in the last iteration or from the beginning of the execution of the algorithm (using one or the other varies among ACO variants, some of them, as $\mathcal{MM}$AS, include both options). The function $F(\cdot)$ is specific to the considered problem and is used to compute the amount of pheromone to be deposited on the solution components.

Variant	**Transition rule**	**Local update**	**Global update**
AS	*random proportional rule*: $p_j^k = \frac{[\tau_j]^\alpha \cdot [\eta_j]^\beta}{\sum_{h \in N^f} [\tau_h]^\alpha \cdot [\eta_h]^\beta}$	*ant density*: $\tau_j = \tau_j + Q_1$, *ant quantity*: $\tau_j = \tau_j + (Q_2 \cdot \eta_j)$, where Q_1 and Q_2 are constants	*ant cycle*: $\tau_j = (1-\rho) \cdot \tau_j + \sum_{k=1}^{m} \Delta\tau_j^k$, where $\Delta\tau_j^k = \begin{cases} F(k) & \text{if } j \in s^k \\ 0 & \text{otherwise} \end{cases}$
Ant-Q	*pseudo-random proportional rule*: $p_j^k = \begin{cases} \arg\max_{h \in N^f} \{\tau_h^\alpha \cdot \eta_h^\beta\} & \text{if } q \leq q_0 \\ \text{same as in AS} & \text{otherwise} \end{cases}$, where q is a uniformly distributed random value	*local reinforcement*: $\tau_j = (1-\alpha) \cdot \tau_j + \alpha \cdot \left[\Delta_{\tau_j} + \gamma \cdot \max_{h \in N^f} \tau_h\right]$, where Δ_{τ_j} is defined as: $\frac{W}{cost}$, W is a constant and *cost* is the solution's cost.	*global reinforcement*: $\tau_j = (1-\alpha) \cdot \tau_j + \alpha \cdot \left(\Delta\tau_j^{best} + \gamma \cdot \max_{h \in N^f} \tau_h\right)$, where $\Delta\tau_j^{best}$ is defined as in AS.
$\mathcal{MM}$AS	*random proportional rule*: same as in AS	—	*pheromones update rule*: $\tau_j = \max\{\tau_{min}, \min\{\tau_{max}, (1-\rho) \cdot \tau_j + \Delta\tau_j^{best}\}\}$, where $\Delta\tau_j^{best} = \begin{cases} F(s^{best}) & \text{if } j \in s^{best} \\ 0 & \text{otherwise} \end{cases}$

Table 5.1 Continued.

Variant	Transition rule	Local update	Global update
ANTS	*additive random proportional rule*: $p_j^k = \frac{\alpha \cdot \tau_j + [1-\alpha] \cdot \eta_j}{\sum_{h \in N^f} \alpha \cdot \tau_h + [1-\alpha] \cdot \eta_h}$	—	*trail update*: $\tau_j = \tau_j + \Delta\tau_j^k$, where $\Delta\tau_j^k$ is defined as: $\tau_0 \cdot \left(1 - \frac{z_{curr} - \mathrm{LB}}{\bar{z} - \mathrm{LB}}\right)$, $\bar{z}$ is the average cost of the solutions, z_{curr} is the cost of the current solution and LB is a lower bound for the problem solution cost
ACS	*pseudo-random proportional rule*: $p_j^k = \begin{cases} \arg\max_{h \in N^f} \{\tau_h^\alpha \cdot \eta_h^\beta\} & \text{if } q \leq q_0 \\ \text{same as in AS} & \text{otherwise} \end{cases}$ where q is a uniformly distributed random value	*local pheromone update rule*: $\tau_j = (1-\varphi) \cdot \tau_j + \varphi \cdot \tau_0$, where τ_0 is the pheromone lower bound	*global pheromone trail updating rule*: $\tau_j = \begin{cases} (1-\rho) \cdot \tau_j + \rho \cdot \Delta\tau_j^{best} & \text{if } j \in s^{best} \\ \tau_j & \text{otherwise} \end{cases}$, where $\Delta\tau_j^{best} = F(s^{best})$
IWD	*random selection rule*: $p_j^k = \frac{\frac{1}{\epsilon + g(soil_j)}}{\sum_{h \in N^f} \left(\frac{1}{\epsilon + g(soil_h)}\right)}$	*local soil update*: $soil_j = (1-\varphi) \cdot soil_j - \varphi \cdot \Delta soil_j$, where $\Delta soil_j$ is defined in Equations 5.7	*global soil update*: $soil_j = \begin{cases} (1+\rho) \cdot soil_j - \rho \cdot \Delta soil_j^{best} & \text{if } j \in s^{best} \\ soil_j & \text{otherwise} \end{cases}$, where $\Delta soil_j^{best} = F(s^{best})$

decrease. In other words, in IWD, the best solution components are characterized by low soil values, whereas in ACO they are characterized by large pheromone values. It is important to clarify this difference, as the comparison of these two concepts (pheromones and soil) might be prone to confusion. In practice, what is important to remember is that the construction of solutions in ACO is biased towards higher pheromone values, while in IWD it is biased towards lower soil values. Additionally, in IWD, soil can become negative in high quality solution components, while in ACO pheromones are strictly positive.

The other concept that deserves some attention is the water drop's *velocity*. This concept does not exist in ACO and, as mentioned in Section 5.2, it comes from the metaphoric idea that water drops move with certain velocity and remove soil from the riverbed. In IWD, the velocity vel^k of a water drop k is used to compute the amount of soil $\Delta soil_j$ that the water drop collects when adding a new solution component j and it is updated according to Equation 5.1:

$$vel^k = vel^k + \frac{a_v}{b_v + c_v \times [soil_j]^2}, \tag{5.1}$$

where a_v, b_v, c_v are user selected parameters.

Water drops that select "good" solution components (i.e., components that have a low soil value) tend to be faster, and therefore, a water drop's velocity somehow measures the quality of the partial solution under construction. A water drop's velocity determines the extent to which the soil will decreased after a solution component is added to a water drop's partial solution: faster water drops remove more soil from the added solution components than slower water drops. As a consequence, since solution components with less soil have a higher probability of being selected by another water drop, velocity is also a way to control the exploration-exploitation capabilities of the algorithm. This can be done, for example, by selecting the initial value of the water drops' velocity. If a low initial velocity is chosen, water drops will tend to have a more exploratory behavior, while if a high initial velocity is chosen they will tend to exploit more the soil information.

In the next three subsections we will compare the *stochastic solution construction*, the *local update* and *global update* metaheuristic components used in IWD with those used in some of the ACO variants proposed in the literature. In particular, we will show that:

- the *random selection rule* used by IWD is a simplification of the *random*

proportional rule rule implemented in Ant System (Dorigo 1992b; Dorigo et al. 1991a, 1996);

- the *local soil update* of IWD is a special cases of Ant-Q's *local reinforcement* (Gambardella and Dorigo 1995); and
- the *global soil update* of IWD is a special case of Ant Colony System's *global pheromone trail updating rule* (Dorigo and Gambardella 1996, 1997b; Gambardella and Dorigo 1996).

5.3.1 Stochastic Solution Construction

Ants construct solutions by adding new components probabilistically chosen using a function of the pheromone values and of the heuristic information. We refer to this function as *transition rule* (see second column of Table 5.1). The transition rule not only states which information will be used by ants to choose the next solution component, but also how the relative importance of such information will be weighted. For example, in the transition rule of AS (Dorigo et al. 1996), the weighting strategy consists in using two parameters α and β that modulate the value of τ and η, respectively; in ANTS (Maniezzo and Carbonaro 2000), a parameter $\alpha \in [0,1]$ allows to change the relative importance of τ and η in the transition rule (see Table 5.1). Equation 5.2 and 5.3 show the transition rules used in Ant System and in IWD:

$$p_j^k = \frac{[\tau_j]^\alpha \cdot [\eta_j]^\beta}{\sum_{h \in N^f} [\tau_h]^\alpha \cdot [\eta_h]^\beta}, \tag{5.2}$$

$$p_j^k = \frac{\frac{1}{\epsilon + g(soil_j)}}{\sum_{h \in N^f} (\frac{1}{\epsilon + g(soil_h)})}, \tag{5.3}$$

where N^f is the set of feasible solution components that can still be added to the partially built solution, $j \in N^f$ is a solution component in the search space and k is one of the m ants/water drops building a solution, and ϵ, in Equation 5.3, is a small positive constant used to avoid a possible division by zero.

It is easy to see that by setting $\alpha = -1$ and $\beta = 0$ in the transition rule of Ant System, it becomes the same one used in IWD. Note, however, that transition rule in IWD only includes the information given by the soil (i.e., heuristic information is not used). Additionally, because the value of soil can become negative in the solution components, IWD applies a function $g(\cdot)$ to

$soil_j$ so that its value in Equation 5.3 remains positive:

$$g(soil_j) = \begin{cases} soil_j & \text{if } \min\limits_{h \in N^f} soil_j \geq 0, \\ soil_j - \min\limits_{h \in N^f} soil_j & \text{otherwise.} \end{cases} \tag{5.4}$$

In both ACO and IWD, the initial value of pheromone/soil, as well as other parameters, such as m, the number of ants/water drops, or the value of α, β, etc., are user selected parameters that have to be chosen according to the problem considered.[7]

5.3.2 Local Update

The local pheromone update allows ants to update the pheromones not only after having built a complete solution, but also while constructing it.

An ACO variant implementing local pheromone update is Ant-Q (Gambardella and Dorigo 1995). In Ant-Q's local pheromone update, pheromones are updated immediately after a component is added to a partial solution using the formula shown in Equation 5.5. Comparing Equation 5.5 with IWD's local soil update given in Equation 5.6, we can see that the two updates are very similar:

$$\tau_j = (1 - \alpha) \cdot \tau_j + \alpha \cdot \left[\Delta_{\tau_j} + \gamma \cdot \max_{h \in N^f} \tau_h\right] \tag{5.5}$$

$$soil_j = (1 - \varphi) \cdot soil_j - \varphi \cdot \Delta soil_j^k \tag{5.6}$$

In particular, if we set the value of $\gamma = 0$ in Equation 5.5, the two equations become virtually identical. However, while α, Δ_{τ_j}, γ, and φ are fixed parameters, the value of $\Delta soil_j^k$ in IWD has to be computed using Equation 5.7, involving the velocity vel^k of the water drop and the *heuristic undesirability* (HUD_j) of the solution component j that is being added. $\Delta soil_j^k$ is computed for every water drop after a solution component is added to the partial solution the water drop is constructing. First the water drop k updates its velocity vel^k according to Equation 5.1 and then $\Delta soil_j^k$ is computed using Equation 5.7:

[7] Finding values for the parameters of stochastic algorithms that guarantee a good algorithm performance is known to be a non-trivial task. See (Stützle et al. 2012) for a comprehensive review of how this problem has been studied in the ACO literature.

$$\Delta soil_j^k = \frac{a_s}{b_s + c_s \cdot [HUD_j / vel^k]^2}, \tag{5.7}$$

where a_s, b_s, c_s are user selected parameters. Therefore, the value of $\Delta soil_j^k$ tends to be larger for solution components with lower soil (because of the velocity update of Equation 5.1) and for those with low HUD_j. Parameter b_v, in Equation 5.1, and parameter b_s, in Equation 5.7, are used to avoid a possible division by zero.

As we discussed in Section 5.3, water drops' velocity can be seen as an indicator of the quality of the partial solution constructed so far, that is, faster water drops have added components with lower soil. However, computing the desirability of a solution component in terms of the velocity (quality of a partial solution) and of the heuristic undesirability, as is defined for $\Delta soil_j^k$, is very similar to the abandoned idea of *ant quantity* (see AS local update procedure in Table 5.1).

As explained in Section 5.2, each water drop k memorizes the amount of soil collected from the solution components added to the solution that it is constructing in a variable called $collected\_soil^k$. The new value of $collected\_soil^k$ is computed by adding the value of $\Delta soil_j^k$ to its current value (which contains the amount of soil collected from previous solution components), as it is shown in Equation 5.8:

$$collected\_soil^k = collected\_soil^k + \Delta soil_j^k, \tag{5.8}$$

Last, one may also ask if the inspiring metaphor of "intelligent water drops removing soil from the riverbed" is a realistic model of the process of erosion in rivers. For example, if soil is removed, it is unclear why then the new amount of soil is computed by an equation such as Equation 5.6 that uses a decay factor φ, and not simply by subtracting $\Delta soil_j^k$. Additionally, the metaphor of water drops acting as individual particles removing the soil in the riverbeds is unrealistic, to say the least, since water in a river should rather be seen, and studied, as a moving fluid. In this sense, if the goal of the author was to test the optimization capabilities of rivers formation (as it is mentioned repeatedly in (Shah-Hosseini 2007, 2008, 2009)), it would have been a better approach to start with some of the models available in the scientific literature describing this process (e.g., Merritt et al. 2003).

5.3.3 Global Update

The global pheromone update in ACO is performed at the end of an iteration once all solutions have been completed. The main goal of this component is to give a *positive feedback* by increasing the amount of pheromone associated to solution components that belong to good solutions; common choices in ACO variants are to update pheromones that belong to the components of the best solution s^{best} found in the current iteration (*iteration-best* update) or since the first iteration of the algorithm (*global-best* update), but other options have been examined too (Dorigo and Stützle 2004). Solution components that receive a higher amount of pheromone will have a higher probability of being selected by other ants in the next iterations.

The global update component in ACS[8] and in IWD[9] is defined, respectively, as follows:

$$\tau_j = \begin{cases} (1-\rho)\cdot\tau_j + \rho\cdot\Delta\tau_j^{best} & \text{if } j \in s^{best} \\ \tau_j & \text{otherwise} \end{cases}, \tag{5.9}$$

$$soil_j = \begin{cases} (1+\rho)\cdot soil_j - \rho\cdot\Delta soil_j^{best} & \text{if } j \in s^{best} \\ soil_j & \text{otherwise} \end{cases}, \tag{5.10}$$

where the parameter $\Delta\tau_j^{best}$ is commonly defined as the inverse of the total cost of the solution s^{best}, while $\Delta soil_j^{best}$ is proportional to the soil collected by the best water drop divided by the number of solution components:

$$\Delta soil_j^{best} = collected\_soil^{best} / N^{best} - 1 \tag{5.11}$$

The similarity between the two equations is clear. Equation 5.9 easily converts into Equation 5.10 by multiplying ρ by -1. However, a more formal way to see this is via a redefinition of the interval over which the parameter ρ can vary in Equation 5.9. That is, if we change this interval from its typical value of $[0,1]$ to $[-1,0]$, we also convert Equation 5.9 into Equation 5.10. Because of

[8] ACS is one of the oldest and best performing ACO variants (Dorigo and Gambardella 1997a); its global update rule is called *global pheromone trail updating rule*.

[9] There are two versions of this component in IWD. In the first one (Shah-Hosseini 2007) the ρ parameter was defined in $[0,1]$, making Equation 5.10 and 5.9 identical. However, for unknown reasons, in a later publication (Shah-Hosseini 2009) the interval of variability of parameter ρ was changed to $[-1,0]$, leading to a somewhat different behavior of the global update procedure, as explained here.

this, the *global soil update* component is a special case of the *global pheromone trail updating rule* proposed in Ant Colony System by Dorigo and Gambardella (1996).

The global soil update, as defined in (Shah-Hosseini 2009), has different outcomes depending on the value of ρ and $soil_j$ in the solution component. For simplicity, let us consider first the second summand in the first case of Equation 5.10, that is, $-\rho \cdot \Delta soil_j^{best}$. Because $\Delta soil_j^{best}$ is defined as always positive (see Equation 5.7) and as we have it multiplied by $-\rho$, the result of this second summand will always be negative and contribute with a *positive feedback* to the solution component, that is, a decrease in the value of *soil*.

On the other hand, the type of feedback given by the first summand in the first case of Equation 5.10, $(1+\rho) \cdot soil_j$, it is more difficult to understand. It is easy to see that if $soil_j < 0$ the product $(1+\rho) \cdot soil_j$ will be negative, and therefore, this summand contributes with a *positive feedback* to the solution component, which is the desired behavior (i.e., removing soil increases the probability that future water drops will select the component). However, if $soil_j > 0$, the resulting value of this summand will be positive, and therefore, it contributes with a *negative feedback* to the solution component and the function of the update in this case is just the opposite of what it should be.

5.4 Modifications of IWD

Very often, after a new metaheuristic is published, different modifications are proposed to enhance its performance and/or to overcome its drawbacks. In this section, we review the literature on IWD with a particular focus on the improvements that have been proposed since its initial publication in 2007. We provide compelling evidence that all these improvements were already present in ACO variants. To select the relevant literature, we searched for the string "intelligent water drop" in the title or in the abstract of the articles indexed in Scopus (`www.scopus.com`) and Google Scholar (`http://scholar.google.com`). From this set of articles we selected all those published in journals and those published in conferences that included at least one variant of the original IWD metaheuristic components. The final set consisted of 7 articles which are presented and discussed in the following.

1. Duan et al. (2008, 2009) were the first to propose a variant of IWD where problem specific information is added to IWD's random selection rule.

This is a relatively minor modification that, in the case of ACO, was already present in its very first formulation (Dorigo et al. 1991a). Later, Booyavi et al. (2014) and Teymourian et al. (2016b) have proposed the use of a parameter λ to weight the importance of soil with respect to heuristic information. Also this weighting mechanism is part of most ACO variants, including some of very first ones (Dorigo and Stützle 2004).

2. Niu et al. (2012) proposed five modifications to enhance the original IWD implementation; these are: *random soil and velocity initialization*, *conditional probability computation*, *bounded local soil update*, *elite global soil update*, and *combined local search*.

   - *Random soil and velocity initialization* aim is to improve the diversity of the initial solutions of the algorithm and, according to the authors, helps avoiding premature convergence. Unfortunately, the authors did not test this hypothesis, for example, by comparing the random initialization with the scheme originally proposed for IWD.
   - *Conditional probability computation* consists of two changes in the stochastic solution construction procedure: (i) to include the heuristic information in the random selection rule along with a parameter to weight its relative importance; and (ii) to select the lower-cost component (i.e., greedy selection) with probability φ_0 and to use the random selection rule with the modification described in (i) otherwise. These two modifications were already proposed in the context of ACO. The use of a parameter to weight the relative importance of the heuristic information, modification (i), is part of the *random proportional rule* of Ant System (see AS transition rule in Table 5.1), the first ACO algorithm ever published (Dorigo 1992b; Dorigo et al. 1991a, 1996); while modification (ii) was used in the *pseudo-random proportional rule* of Ant Colony System (see ACS transition rule in Table 5.1) and was first introduced in (Gambardella and Dorigo 1996).
   - *Bounded local soil update* uses two values, Δ_{max} and Δ_{min}, to set, respectively, the maximum and minimum change in the amount of *soil* for a given solution component. This very same idea, in the form of upper and lower bounds to the value of pheromones, was introduced in the ACO variant called $\mathcal{MAX}$–$\mathcal{MIN}$ Ant System, first proposed in a Technical Report in 1996 (Stützle and Hoos 1996), and later published

in (Stützle and Hoos 2000).

- *Elite global soil update* uses more than one water drop to update the soil in the *global soil update*. The idea of using more than one solution to update the pheromone trails was also explored in the context of ACO. This was done with Elitist ant system, first proposed in Dorigo's PhD thesis (Dorigo 1992b) and then published in (Dorigo et al. 1996).
- *Combined local search* adds a local search phase to the IWD algorithm. As said when describing the ACO metaheuristic in Section 5.1, *daemon actions* often consist of a local search that improves the solutions constructed by the ants. The first publications to introduce local search in ACO algorithms are (Dorigo and Gambardella 1997a; Stützle and Hoos 1997); these were followed by many other (e.g., Gambardella et al. 1999; Maniezzo and Colorni 1999) and nowadays, the usage of a local search routine is pretty standard in the best performing ACO algorithms (Dorigo and Stützle 2004).

3. Msallam and Hamdan (2011) propose to reinitialize the soil and the velocities of all water drops after reaching a certain number of iterations without improving the global best solution. The very same reinitialization scheme proposed in (Msallam and Hamdan 2011) was proposed for $\mathcal{MM}$AS (Stützle and Hoos 1997, 2000) and has been widely used in the ACO literature (Dorigo and Stützle 2004).

4. Alijla et al. (2014) propose to replace the original (i.e., Equation 5.3) random selection rule of IWD with two ranking selection methods, one linear and one exponential. Both selection methods rank in descending order the feasible solution components according to their value of soil. In linear selection, the probability of selecting a solution component is computed using a linear function where a user selected parameter, SP, controls the steepness of the gradient. In exponential selection, the feasible components are weighted exponentially according to their ranks. These two ranking selection methods aim to overcome three shortcomings of IWD's random selection rule, that is its inability (i) to accommodate negative soil values, (ii) to differentiate between solution components with small soil difference, and (iii) to control local stagnation. Unlike IWD, ACO algorithms do not present these shortcomings because pheromones

have always positive values, which avoids shortcoming (i), and there are mechanisms to bound pheromones lower and upper limit, avoiding the shortcomings (ii) and (iii).

Although the authors argue that these problems are caused by the way in which the transition rule has been defined in IWD, shortcomings (ii) and (iii) are rather the result of having negative soil values and of the way in which global soil update is defined. That is, IWD has mechanisms to manage solution components with negative and positive values in the transition rule (see Equation 5.3 and 5.4); however, this is not the case for the global soil update (see Equation 5.10) (see the discussion on the global soil update in Section 5.3.3).

In this review, we have identified the ideas proposed to improve the performance of IWD since its initial publication in 2007. A few articles seek to overcome drawbacks and limitation of the metaheuristic (Alijla et al. 2014; Msallam and Hamdan 2011), while others propose modifications to enhance its exploration-exploitation capabilities (Booyavi et al. 2014; Duan et al. 2008, 2009; Niu et al. 2012; Teymourian et al. 2016a). Although the research on IWD is not particularly rich, we found that almost everything proposed to modify this metaheuristic consists of ideas that were first proposed in the context of ACO. In fact, most of these modifications can be matched directly to well-know ACO variants that have been in the literature for many years, even before the first IWD algorithm was proposed.

5.5 Summary

In this chapter, we compared the "novel" *intelligent water drops* metaheuristic to ideas previously proposed in the context of ACO. The result from this analysis shows that IWD is a particular case of ACO. In particular, the random selection rule of IWD is a simplified version of the random proportional rule of ant system, the very first ACO algorithm. The local soil update in IWD is a special case of the local reinforcement that was proposed in the Ant-Q algorithm, where the only minor difference between IWD and this early ACO variant is the parameter $\Delta soil_j$, which is computed using the water drop's velocity and the solution components' $soil_j$ and heuristic undesirability. Last, the global soil update of IWD is also a special case of an ACO variant, in this case the global pheromone trail updating rule proposed in Ant Colony System, in which the

parameter interval of ρ (that in ACO is $[0, 1]$) is redefined to the interval $[-1, 0]$. In addition to studying the metaheuristic components proposed in IWD, we briefly analyzed the rationale behind the definition of $\Delta soil_j$ and the general idea over which the local soil update is based. Our conclusion is that the metaphor of "intelligent water drops removing soil from the ground of a river" is based on unrealistic assumptions on how erosion works in river systems.

From the review of the literature of IWD, we found that most of the research done on IWD is just a repetition of research ideas that had been explored in the context of ACO. Indeed, with only one exception, all the modifications proposed for the "novel" IWD metaheuristic have a direct correspondence with a specific modification proposed many years before for an ACO algorithm.

Chapter 6

"Novel" Metaheuristics for Continuous Optimization

In this chapter, we present a rigorous, component-based analysis of the *grey wolf optimizer* (Mirjalili et al. 2014), the *moth-flame algorithm* (Mirjalili 2015a), the *whale optimization algorithm* (Mirjalili and Lewis 2016), the *firefly algorithm* (Yang 2009), the *bat algorithm* (Yang 2010), the *antlion optimizer* (Mirjalili 2015b), and the *cuckoo search* (Yang and Deb 2009), which are among the most widespread "novel" metaphor-based metaheuristic algorithms[1] published in the literature. These algorithms were chosen from the Evolutionary Computation-bestiary (EC-bestiary) (Campelo and Aranha 2021a) using as sole criteria that they were proposed for the approximate solution of continuous optimization problems and that they were highly-cited (data from Google Scholar retrieved on June 12, 2023, shows the following citation counts: *grey wolf optimizer*: 11,841 citations; *moth-flame algorithm*: 3,241 citations; *whale optimization algorithm*: 8,034 citations; *firefly algorithm*: 4,869 citations; *bat algorithm*: 6,003 citations; *antlion optimizer*: 2,691 citations; and *cuckoo search*: 7,729 citations.)

In addition to deconstructing the seven algorithms into their components and relating them with equivalent components proposed in well-established techniques, such as *particle swarm optimization* and *evolutionary algorithms*, we analyze the use of the metaphors that inspired these algorithms to understand whether their usage has brought any novel and useful concepts to the field. Thus, for each of the metaphors used in these algorithms, we evaluate whether the following criteria are fulfilled:

[1] In this chapter, we may also refer to the studied metaheuristics simply as algorithms, since they already have the words "algorithm" and "optimizer" in their names.

- **Usefulness** – Does the metaphor bring useful ideas on how to solve optimization problems?
- **Novelty** – Were the ideas brought by the use of the new metaphor novel in the field of metaheuristics when they were proposed?

The main finding of our analysis is that, despite all these algorithms were presented as *novel* approaches, none of them proposes any new ideas; rather, their authors used variations of the same algorithm components that have been for years in the literature of metaheuristics, framed them into new metaphors and published them as new. In particular, we found that the *grey wolf*, *moth-flame*, *whale*, *firefly* and *bat* algorithms use concepts that are the same as those previously used in the context of *particle swarm optimization*, while the *antlion* and *cuckoo* algorithms use those of *evolution strategies* and *differential evolution*.

From the analysis of the metaphors that inspired the seven algorithms, we found that they cannot be used to explain the majority of the design choices in them. The description of the metaphors in the articles are vague for the most part; important aspects are not mentioned at all or lack sufficient detail (such as, what exactly is being optimized in the behavior that inspired the algorithm), while those that are irrelevant for the purpose of designing an optimization algorithm are abundant and overemphasized (e.g., how many species of fireflies/moths/whales exist and how amazing/fancy/unique they are in the opinion of the authors). Additionally, we observed that it is common practice for the authors of this kind of publications to relate metaphors and mathematical models using (i) ideas that do not belong to the metaphor originally described, and/or (ii) ideas that radically change the behavior that reputedly inspired the algorithm.

6.1 Overview of Particle Swarm Optimization

Particle swarm optimization (PSO) (Kennedy and Eberhart 1995) is a population-based algorithm proposed for the approximate solution of continuous optimization problems. In PSO, a swarm of particles, each representing a solution to the problem at hand, try to identify the most promising areas of the search space by applying a set of rules that take into account the locations of good solutions that they and their neighboring particles have visited in the past. To do so, each particle i knows, at every iteration t, its current position $\vec{x}_t^i$, velocity $\vec{v}_t^i$, and personal best position $\vec{p}_t^i$, as well as the best position $\vec{l}_t^i$ of the best particle in

its neighborhood. The goal of using vectors $\vec{p}_t$ (called *cognitive* influence) and vectors $\vec{l}_t$ (called *social* influence) is to combine the knowledge acquired by each particle during the search with the knowledge of the best-informed individual in the neighborhood of the particle.

Depending on the way the neighborhood is defined in the algorithm, it is possible to create many different population topologies. For example, if the neighborhood of a particle consists of the two adjacent (i.e., closest) particles we have the so-called *ring* topology, while assigning all particles to the neighborhood of all other particles creates a *fully-connected* or *g*best topology. In the latter, the local best particle is called the *global best* and its position is indicated by $\vec{g}_t$. When using the ring topology, the information about where the best-so-far solution is located spreads slowly among particles, while with the fully-connected or *g*best topology the entire swarm knows immediately the position of the best-so-far solution at each iteration. Along with these two topologies, many others have been studied in the literature, including *wheels*, *lattices*, *stars*, and *randomly assigned edges* (Mendes et al. 2004).

Since its initial publication, PSO has been extensively studied and applied to many problems, resulting in a plethora of variants that range from little refinements of the original algorithm to new versions of the algorithm that contain quite elaborate changes and novel ideas. In the remaining of this section, we present a number of PSO variants that are relevant to our study.

- *Standard PSO (Shi and Eberhart 1998, 1999) — StdPSO*. In StdPSO, particles update their positions using the following rule:

$$\vec{x}^i_{t+1} = \vec{x}^i_t + \vec{v}^i_{t+1}, \tag{6.1}$$

$$\vec{v}^i_{t+1} = \omega\vec{v}^i_t + \varphi_1\vec{a}^i_t \odot (\vec{p}^i_t - \vec{x}^i_t) + \varphi_2\vec{b}^i_t \odot (\vec{l}^i_t - \vec{x}^i_t), \tag{6.2}$$

  where ω is called inertia weight and controls the effect of the velocity vector at time t, φ_1 and φ_2 are called acceleration coefficients and weigh the relative importance given to the cognitive and social influence, $\vec{a}^i_t$ and $\vec{b}^i_t$ are two random vectors[2] used to provide diversity to the particles' movement, and $\odot$ indicates the Hadamard (entrywise) product between two vectors.

[2]Note that, in the formulation of the velocity update rule of PSO shown in Equation 6.2, vectors $\vec{a}^i_t$ and $\vec{b}^i_t$ are used instead of the random diagonal matrices U^i_{1t} and U^i_{2t}. While both formulations are equivalent in practice, the use of the formulation in Equation 6.2 makes easier to understand the comparisons that we make in this chapter.

- *Standard PSO 2011 (Clerc 2011; Zambrano-Bigiarin et al. 2013) — SPSO-2011.* This variant is a modified version of StdPSO that prevents the issue of rotation variance. The velocity update rule of StdPSO was revised in SPSO-2011 as follows:

$$\vec{v}_{t+1}^{\,i} = \omega \vec{v}_t^{\,i} + \vec{x}_t'^{\,i} - \vec{x}_t^{\,i}, \tag{6.3}$$

where $\vec{x}_t'^{\,i}$ is a randomly generated point in the hypersphere $\mathcal{H}_i(\vec{c}_t^{\,i}, |\vec{c}_t^{\,i} - \vec{x}_t^{\,i}|)$ with center $\vec{c}_t^{\,i}$ and radius $|\vec{c}_t^{\,i} - \vec{x}_t^{\,i}|$, and $|\cdot|$ indicates the vector's L2 norm. The computation of the center $\vec{c}_t^{\,i}$ is defined as follows:

$$\vec{c}_t^{\,i} = (\vec{L}_t^{\,i} + \vec{P}_t^{\,i} + \vec{x}_t^{\,i})/3 \tag{6.4}$$

where

$$\begin{aligned} \vec{P}_t^{\,i} &= \vec{x}_t^{\,i} + \varphi_1 \vec{a}_t^{\,i} \odot (\vec{p}_t^{\,i} - \vec{x}_t^{\,i}) \\ \vec{L}_t^{\,i} &= \vec{x}_t^{\,i} + \varphi_2 \vec{b}_t^{\,i} \odot (\vec{l}_t^{\,i} - \vec{x}_t^{\,i}) \end{aligned}. \tag{6.5}$$

- *Fully informed PSO (Mendes et al. 2004) — FiPSO.* In FiPSO, the velocity update rule is as follows:

$$\vec{v}_{t+1}^{\,i} = \chi\Big(\vec{v}_t^{\,i} + \sum_{k \in T_t^i} \varphi \vec{a}_{kt}^{\,i} \odot (\vec{p}_t^{\,k} - \vec{x}_t^{\,i})\Big), \tag{6.6}$$

where $\chi = 0.7298$ is a constant value called constriction coefficient (Clerc and Kennedy 2002), T_t^i is the set of particles in the neighborhood of i, and φ is a parameter. As opposed to the existing PSO variants that use the *best-of-neighborhood* model, in which the *social* influence of a particle comes from either $\vec{l}_t^{\,i}$ or $\vec{g}_t$, in FiPSO, a particle is influenced by all of its neighbors. The *social* influence model proposed in FiPSO is referred to as *fully informed* in the literature of PSO.

- *Simple dynamic particle swarms (Peña 2008a,b) — SDPSs.* This class of PSO algorithms do not make use of the randomness induced by vectors $\vec{a}_t$ and $\vec{b}_t$ in the position update rule of the particles. Also, they are often implemented without a velocity vector $\vec{v}_t^{\,i}$. Most SDPSs can be instantiated from the following generalized position update rule:

$$\vec{x}_{t+1}^{\,i} = \vec{x}_t^{\,i} + \epsilon(\vec{y} - \vec{x}_t^{\,i}), \tag{6.7}$$

where ϵ is a parameter and $\vec{y}$ is a vector obtained by combining the information of two or more particles in the swarm. Examples of how vector $\vec{y}$ can be computed in SDPSs include:

$$\text{Standard: } \vec{y} = \frac{u_1 \vec{x}_t'^1 + u_2 \vec{x}_t'^2}{u_1 + u_2}, \tag{6.8}$$

$$\text{Normal: } \vec{y} = \mathcal{N}(\frac{\vec{x}_t'^1 + \vec{x}_t'^2}{2}, |\vec{x}_t'^1 - \vec{x}_t'^2|), \tag{6.9}$$

where $\vec{x}_t'^1$ and $\vec{x}_t'^2$ are position vectors chosen according to some criterion, and u_1 and u_2 are two real parameters whose value is typically set in the range $(0, 1]$. SDPSs algorithms vary in all kind of aspects, including the number of particles participating in the computation of $\vec{y}$, which can be from 1 to all particles and the way in which the current position of a particle is taken into account in the computation of $\vec{x}_{t+1}^i$.

- *Extrapolation PSO (Arumugam et al. 2007, 2009) — ePSO.* In *e*PSO, particles do not have a *cognitive* influence component (i.e., vector $\vec{p}_t$) and parameters φ_{1t} and φ_{2t} are replaced by two so-called "extrapolation" coefficients. The position update rule proposed for *e*PSO is as follows:

$$\vec{x}_{t+1}^i = \vec{g}_t + \varphi_{1t}\vec{g}_t + \varphi_{2t}(\vec{g}_t - \vec{x}_t^i), \tag{6.10}$$

where $\varphi_{1t} = \mathcal{U}[0,1]\, k_1$, $\varphi_{2t} = k_1 e^{k_2 \Lambda_t^i}$, $k_1 = k_2 = e^{-t/t_{max}}$, $\Lambda_t^i = |(f(\vec{g}_t) - f(\vec{x}_t^i))/f(\vec{g}_t)|$, and $f(\cdot)$ refers to the objective function of a minimization problem. This position update rule combines the information of the global best solution ($\vec{g}_t$) with the current position of the particles ($\vec{x}_t^i$) and adjusts the displacement of the particle in terms of the difference between $f(\vec{g}_t)$ and $f(\vec{x}_t^i)$. Because of the way φ_{1t} and φ_{2t} are computed, a particle will experience a strong attraction towards $\vec{g}_t$ when its quality is much lower than that of $f(\vec{g}_t)$, and a weak attraction towards $\vec{g}_t$ when its quality is similar to $f(\vec{g}_t)$.

6.2 Overview of Evolutionary Computation

As we outlined in Chapter 2, the field of evolutionary computation is vast and includes several approaches, such as *evolution strategies* (Rechenberg 1971; Schwefel 1977), *genetic algorithms* (Goldberg 1989; Holland 1975), *genetic program-*

ming (Koza 1992) and *differential evolution* (Storn and Price 1997). In this section, we overview the main concepts of evolution strategies and differential evolution, which are the two metaheuristics that are more relevant for the analyses that we present in the following sections.

6.2.1 Evolution Strategies

Evolution Strategies (ESs) (Bäck et al. 1991; Bäck and Schwefel 1993; Rechenberg 1971, 1973; Schaffer 1985; Schwefel 1981) are some of the best-known evolutionary algorithms proposed to deal mainly with continuous optimization problems. In ESs, as in the rest of evolutionary algorithms, the idea is to simulate the process of natural evolution in order to evolve one or several solutions by iteratively applying the operators of *parental selection*, *recombination*, *mutation* and *survival selection* (Michalewicz and Schoenauer 2013). To better illustrate how ESs work, we can consider the classic $(\mu + \lambda)$–ES (Schwefel 1981), which is shown in Algorithm 7.

Algorithm 7 The $(\mu + \lambda)$-evolution strategy

```
1:  begin
2:      t ← 0
3:      initialize μ parents (solutions)
4:      evaluate μ parents
5:      while not termination-condition do
6:          t ← t + 1
7:          apply recombination to μ parents to create λ offspring (new solu-
                tions)
8:          apply mutation to offspring
9:          evaluate offspring
10:         select μ solutions from the set of (μ-parents + λ-offspring) and use
                them as parents for the next iteration
11:     end while
12: end
```

The $(\mu + \lambda)$–ES is an algorithm in which a population of μ solutions (parents) produce λ new solutions (offspring) to generate a population of $\mu + \lambda$ individuals. The population is then reduced again to μ solutions that constitute the next generation. As we show in Algorithm 7, in order to instantiate a $(\mu + \lambda)$–ES, it is necessary to choose the specific operators to be used. In the following, we describe the operators proposed for the $(\mu + \lambda)$–ES.

- Parental selection. It refers to the way individuals that will be used to generate a new set of solutions are selected; the selection can be deterministic (one or more specific individuals) or probabilistic (individual are selected based on a probability distribution constructed based on their fitness, ranking, etc.). In the $(\mu + \lambda)$–ES, the parental selection operator can be implemented in many different ways (Bäck et al. 1997; Bäck and Schwefel 1993). One option that has been used is to let each parent generate one single offspring at each iteration, which results in $\lambda = \mu$.

- Recombination. The goal of recombination, which is also a type of perturbation, is to create one new solution by combining the information of two or more solutions taken from the current population. Recombination is often regarded as an optional component in the $(\mu + \lambda)$–ES; however, it was used in the early variants of ES and it can be applied in a variety of ways. Some of the most common implementations are discrete, intermediate, global–discrete and global–intermediate recombination (Bäck et al. 1997).

- Mutation. The goal of mutation, which is also a type of perturbation, is to induce small random variations to all the variable encoded in the solutions. ESs generate mutations by sampling a random distribution, such as, the Gaussian distribution, that was used in the original algorithm (Schwefel 1981); the Cauchy distribution, that was introduced in (Kappler 1996) in 1994; and the Lévy distribution introduced in (Iwamatsu 2002; Lee and Yao 2004) in 2002.

- Survival selection. As opposed to parental selection, survival selection refers to the mechanism used to choose the solutions that will be eliminated from the population. In $(\mu + \lambda)$–ES, survival selection operates over parent-offspring couplings, which means that parents will pass from one generation to another until they are replaced by an offspring with better fitness.

6.2.2 Differential Evolution

Differential Evolution (DE) (Price et al. 2005; Storn and Price 1997) is a more recent EC approach proposed to solve continuous optimization problems. Similarly to ESs, in DE, a population of individuals, each one representing a solution to the considered problem, is iteratively improved by applying a number evo-

lutionary operators. DE uses real-valued vectors to represent the individuals, which are indicated as $\vec{x}^i$, for $i = 1, \ldots, n$. The main idea in DE is to use the *mutation* and *recombination* operators to create a new population of solutions, referred to as "trial vectors", so that, at each iteration t, each each individual $\vec{x}_t^i$ is assigned a corresponding trial vector u_t^i. Then, the *survival selection* operator is used to decide which of the two vectors ($\vec{x}_t^i$ or u_t^i) will remain in the population based on its solution quality.

As opposed to most evolutionary algorithms, DE does not make use of a probability distribution in the mutation operator; rather, it uses a geometric approach that is similar to the Nelder-Mead simplex search method to create a new vector, called the *mutant vector*. The *differential mutation* operator, as it is called in the DE terminology, is defined as follows:

$$\vec{m}_t^i = \vec{x}_t^a + \beta \cdot (\vec{x}_t^b - \vec{x}_t^c), \tag{6.11}$$

where i indicates the ith individual in the population, β is a scaling factor, and $\vec{x}_t^a$, $\vec{x}_t^b$ and $\vec{x}_t^c$ are three vectors chosen from the population, such that $\vec{x}_t^a \neq \vec{x}_t^b \neq \vec{x}_t^c$ and $\vec{x}_t^a \neq \vec{x}_t^i$.

The application of the differential mutation operator is followed by the *recombination* operator (Equation 6.12) and then by the *survival selection* operator (Equation 6.13).

$$u_t^{i,k} = \begin{cases} m_t^{i,k}, & \text{if } (\mathcal{U}[0,1] \geq p_a) \vee (k = k_{t,rand}^i) \\ x_t^{i,k}, & \text{otherwise} \end{cases}, \forall k, \tag{6.12}$$

$$\vec{x}_{t+1}^i = \begin{cases} \vec{u}_t^i & \text{if } \vec{u}_t^i \text{ is better than } \vec{x}_t^i \\ \vec{x}_t^i & \text{otherwise} \end{cases}, \tag{6.13}$$

where $k = 1, \ldots, d$ allows to iterate between the values of the vectors, $\mathcal{U}[0,1]$ is a random number sampled from a uniform distribution, p_a is a user-selected parameter in the range $[0,1]$ that controls the fraction of values copied from the mutant vector ($\vec{m}_t^i$) into the trial vector (u_t^i), and $k_{t,rand}^i$ is a randomly chosen dimension that ensures that the trial vector is not a duplicate of the solution $\vec{x}_t^i$. In one iteration of DE, Equation 6.11, 6.12 and 6.13, are applied iteratively to all the individuals in the population until a termination criterion is met.

6.3 Exposing the Grey Wolf, Moth-Flame, Whale, Firefly, Bat, Antlion, and Cuckoo algorithms

In this section, we analyze the *grey wolf*, *moth-flame*, *whale*, *firefly*, *bat*, *antlion* and *cuckoo* algorithms. For each of them, we present (i) the algorithm using the standard optimization terminology, (ii) a component-based comparison with existing techniques, and (iii) the metaphor that inspired the algorithm and discussion on whether it meets the criteria of *novelty* and *usefulness*. For (i) and (ii), we avoid using the vocabulary introduced by the authors of these algorithms because, as we show after presenting each of them, it is unnecessary and misleading in many ways. In our view, one of the main reasons why these algorithms have not been immediately recognized as minor variants of well-established techniques is that they were presented using metaphor-based terminologies that obfuscated their similarities with existing approaches. Therefore, by explaining each algorithm in plain computational terms, we intend to make clearly visible what ideas are being proposed in them and whether they are truly novel or not. Also, we believe that presenting first (i) and (ii) and leaving (iii) at the end allows the reader to better appreciate whether the metaphor contributes at all to the design of the proposed algorithm. Finally, we would like to mention to the reader that all seven algorithms discussed here have publicly available implementations. The ones of the *grey wolf*, *moth-flame*, *whale* and *antlion* can be downloaded from https://seyedalimirjalili.com/ and the ones of the *firefly*, *bat* and *cuckoo* from https://nl.mathworks.com/matlabcentral/profile/authors/2652824.

6.3.1 Grey Wolf Optimizer

The grey wolf optimizer (Mirjalili et al. 2014) (GWO) is an algorithm in which the three iteration-best solutions in the population are used to bias the movement of the remaining solutions. This idea is implemented in GWO by defining three vectors $\vec{s}_t^{\,k}$ (for $k = 1, 2, 3$) as follows:

$$\begin{aligned}
\vec{s}_t^{\,1} &= \vec{x}_t^{\,\text{best1}} - \varphi_t(2\,\vec{r}_t^{\,1} - \vec{1}) \odot \left(2\,\vec{q}_t^{\,1} \odot \vec{x}_t^{\,\text{best1}} - \vec{x}_t^{\,i}\right)^{\text{abs}} \\
\vec{s}_t^{\,2} &= \vec{x}_t^{\,\text{best2}} - \varphi_t(2\,\vec{r}_t^{\,2} - \vec{1}) \odot \left(2\,\vec{q}_t^{\,2} \odot \vec{x}_t^{\,\text{best2}} - \vec{x}_t^{\,i}\right)^{\text{abs}}, \quad \forall i \\
\vec{s}_t^{\,3} &= \vec{x}_t^{\,\text{best3}} - \varphi_t(2\,\vec{r}_t^{\,3} - \vec{1}) \odot \left(2\,\vec{q}_t^{\,3} \odot \vec{x}_t^{\,\text{best3}} - \vec{x}_t^{\,i}\right)^{\text{abs}}
\end{aligned} \tag{6.14}$$

where $\vec{x}_t^{\text{best1}}$, $\vec{x}_t^{\text{best2}}$ and $\vec{x}_t^{\text{best3}}$ are the three best solutions at iteration t, $\vec{r}_t^k$ and $\vec{q}_t^k$ are two random vectors with values drawn from $\mathcal{U}[0,1]$ that induce perturbation to the components of $\vec{s}_t^k$, φ_t is a parameter that decreases linearly from 2 to 0, $\vec{1}$ is a vector of all ones, and $(\cdot)^{\text{abs}}$ indicates the entrywise absolute value of a vector. The entrywise absolute value of a vector is a transformation that can be formally defined as $(\vec{u})^{\text{abs}} = (|u_i|, \dots |u_d|)^{\text{T}}$. The position update rule combining the information of vectors $\vec{s}_t^k$ is defined as follows:

$$\vec{x}_{t+1}^i = (\vec{s}_t^1 + \vec{s}_t^2 + \vec{s}_t^3)/3. \tag{6.15}$$

The Grey Wolf Optimizer is PSO

The mathematical model of GWO is a variant of the one proposed for StaPSO. To better explain how GWO compares to StaPSO, we consider first the mathematical model of GWO without the perturbation component $\varphi_t(2\vec{r}_t^k - \vec{1})$, which is as follow:

$$\begin{aligned}
\vec{s}_t^1 &= \vec{x}_t^{\text{best1}} - \left(2\,\vec{q}_t^1 \odot \vec{x}_t^{\text{best1}} - \vec{x}_t^i\right)^{\text{abs}} \\
\vec{s}_t^2 &= \vec{x}_t^{\text{best2}} - \left(2\,\vec{q}_t^2 \odot \vec{x}_t^{\text{best2}} - \vec{x}_t^i\right)^{\text{abs}}, \ \forall i \\
\vec{s}_t^3 &= \vec{x}_t^{\text{best3}} - \left(2\,\vec{q}_t^3 \odot \vec{x}_t^{\text{best3}} - \vec{x}_t^i\right)^{\text{abs}}
\end{aligned} \tag{6.16}$$

Both StaPSO (Equation 6.5) and GWO (Equation 6.16) are based on the idea of (i) defining, for each particle i in the population, a hyper-triangle in the search space, whose vertices are a function of positions known by i, and (ii) using the centroid of the hyper-triangle in the computation of i's new position. The main difference between the two algorithms is in the way in which the vertices of the hyper-triangle are computed. While in GWO all particles compute the vertices using the same three iteration-best solutions, that is, $\vec{x}_t^{\text{best}\,1}$, $\vec{x}_t^{\text{best}\,2}$ and $\vec{x}_t^{\text{best}\,3}$, in StaPSO, each particle uses its local information, that is, vectors $\vec{x}_t^i$, $\vec{p}_t^i$ and $\vec{l}_t^i$.

In PSO, the goal of using vectors $\vec{v}_t^i$, $\vec{l}_t^i$ and $\vec{p}_t^i$, where $\vec{l}_t^i$ is different for each neighborhood and $\vec{p}_t^i$ is different for each particle, is to include components that allow to balance the relation between *exploration* and *exploitation* of the search space. Adding particles' previous velocity $\vec{v}_t^i$ to their new positions promotes exploration, whereas attracting particles towards known good solutions, such as $\vec{l}_t^i$ and $\vec{p}_t^i$, promotes exploitation. In contrast, in GWO, as seen in Equation 6.16, the entire swarm is attracted towards the same three solutions $\vec{x}_t^{\text{best}\,1}$, $\vec{x}_t^{\text{best}\,2}$ and $\vec{x}_t^{\text{best}\,3}$, which can be useful for intensifying the search in the area defined by these vectors, but prevents particles from exploring other regions.

Not surprisingly, the authors of GWO found that their first version of the

algorithm (which is, as far as it is understood in their article, the one using Equation 6.16) resulted in a poor performing implementation that "is prone to stagnation in local solutions" (Mirjalili et al. 2014, p. 50). To remediate this issue, the authors added a second perturbation component to the computation of vectors $\vec{s}_t^k$ (Equation 6.14), which includes a random vector $\vec{r}_t^k$ multiplied by a linearly decreasing parameter φ_t. This additional perturbation component produces both positive and negative random values in the range $[-2, 2]$, determining a much stronger perturbation to the particles movement than the one initially defined with vector $\vec{q}_t^k$. To avoid particles divergence outside the search space, the impact of $\varphi_t(2\,\vec{r}_t^k - \vec{1})$ in the computation of vectors $\vec{s}_t^k$ is controlled by parameter φ_t whose value decreases linearly from 2 to 0, allowing particles to move closer and closer to $\vec{x}_t^{\text{best}\,1}$, $\vec{x}_t^{\text{best}\,2}$ and $\vec{x}_t^{\text{best}\,3}$ towards the end of the algorithm's execution.

In addition to the computation of vectors $\vec{s}_t^k$, the position update rule of GWO introduced in Equation 6.15 is the same as the computation of the center $\vec{c}_t^i$ in StaPSO—see Equation 6.4. However, in StaPSO, vector $\vec{c}_t^i$ is the center of a hyperspherical distribution from which a random vector is generated, whereas in GWO the computed center becomes the new position of the particle. Because of this difference, GWO's position update rule can also be compared with the standard recombination rule proposed for SDPSs (Equation 6.8) extended to three particles, where vectors $\vec{x}_t'^1, \vec{x}_t'^2$ and $\vec{x}_t'^3$ correspond to vectors $\vec{s}_t^k$ and parameters u_1, u_2 and u_3 are set to 1.

The Metaphor of Grey Wolves Hunting

The authors of GWO say in their original paper (Mirjalili et al. 2014) that they were inspired by the way in which "grey wolves organize for hunting following a strict social hierarchy", where the pack is divided, from top to bottom, in α, β, δ, and ω wolves. According to their description of grey wolves hunting behavior, while α wolves usually take part in hunting and they are in charge of guiding the rest of wolves participating in this activity, the β and δ wolves only take part in hunting occasionally. In GWO, vector $\vec{x}_t^{\text{best}\,1}$ represents the α wolf, $\vec{x}_t^{\text{best}\,2}$ represents the β wolf, $\vec{x}_t^{\text{best}\,3}$ represents the δ wolf, and the rest of solutions in the swarm represent the ω wolves. However, since it is false that the entire pack always participates in hunting every time, saying that solutions $\vec{x}_t^{\text{best}\,1}$, $\vec{x}_t^{\text{best}\,2}$ and $\vec{x}_t^{\text{best}\,3}$ represent the α, β and δ is inaccurate, as it does not follow the description of the "strict social hierarchy" of grey wolves that inspired the

authors when proposing the algorithm.

The authors of GWO mention that there are three phases during hunting, each one composed of a number of steps: (i) *tracking*, *chasing*, and *approaching* the prey; (ii) *pursuing*, *encircling*, and *harassing* the prey until it stops moving; and (iii) *attacking* towards the prey. However, GWO only considers 2 out of the 7 steps mentioned: *encircling*, which is modeled using Equation 6.15, and *attacking*, which is modeled by linearly decreasing the value of φ_t from 2 to 0 in Equation 6.14. In the imagery of the metaphor, when φ_t is lower than 1, wolves concentrate around the prey (therefore attacking it); and when it is greater than 1, they *search* for other prey. Note that, despite *search* is not an activity in the hunting phases of wolves, the authors added this step to the metaphor and explain it as "the divergence among wolves during hunting in order to find a *fitter prey*" (Mirjalili et al. 2014, p. 50).

Based on the description of the "grey wolves hunting" behavior presented by the authors of GWO and the way this behavior is used as a metaphor to develop the proposed algorithm, it is clear that the only contribution of the metaphor is to create confusion and to hide the similarities of the "novel" GWO with PSO. In particular, GWO does not satisfy the criterion of usefulness because there are no components in the behavior of grey wolves that can be used as effective design choices in an optimization algorithm, as evidenced by the fact that the mathematical model originally derived from this behavior resulted in a poor performing technique that stagnated prematurely. Also, as it was shown in the previous section, GWO does not satisfy the novelty criterion because all the algorithm components of GWO correspond to particular cases of algorithm components previously proposed for StaPSO and SDPSs.

6.3.2 Moth-Flame Algorithm

In the moth-flame algorithm (Mirjalili 2015a) (MFA), each solution in the population is assigned a ranking ($rank_t^i$) based on the quality of its current position ($\vec{x}_t^i$), which determines the specific neighbor of the population that will take part in the computation of its position update rule; that is, the solution with $rank_t^i = 1$ will compute its new position using the best overall solution $\vec{g}_t^1$, the one with $rank_t^i = 2$ will use the second best solutions $\vec{g}_t^2$, and so on. Note that the set of vectors $\vec{g}_t$ is the same as the set of vectors $\vec{p}_t$ in PSO, but ordered according to their quality, so that $\vec{g}_t^1$ corresponds to the vector $\vec{p}_t$ with the highest quality and $\vec{g}_t^n$ to the one with the lowest quality. The equation modeling this process

is

$$\vec{x}_{t+1}^{\,i} = \vec{g}_t^{\,rank_t^i} + \varphi_t(\vec{g}_t^{\,rank_t^i} - \vec{x}_t^{\,i})^{abs}, \tag{6.17}$$

with

$$\varphi_t = e^{\delta}\cos(2\,\pi\,\delta), \tag{6.18}$$

where $(\cdot)^{abs}$ denotes the entrywise absolute value, $\delta = (\frac{-t}{t_{max}} - 2)\,\mathcal{U}[0,1] + 1$ and t_{max} is the iteration number at which the algorithm stops. There are two things to note about Equation 6.18. First, the range in which the value of δ is computed spans from $[-1\frac{1}{t_{max}}, 1]$ to $(-2, 1]$ as the value of t grows; second, when $\delta \to -\infty$, the value of $\varphi_t \to 0$. Therefore, the probability of computing large values for φ_t decreases towards the end of the execution of the algorithm, allowing solutions to move closer and closer to their respective vector $\vec{g}_t^{\,rank_t^i}$.

In MFA, every t_{max}/n iterations, the variable $rank_t^i$ of the worst $n - m$ solutions is set to $rank_t^i = m$, where m is computed as $m = \mathsf{round}(n - \frac{t\,(n-1)}{t_{max}})$, n is the population size, and $\mathsf{round}(\cdot)$ indicates the round to the nearest integer function. The goal of doing this is to stop using the $\vec{p}_t$ vector of the $n - m$ solutions to influence the position update of other solutions. Because of the way the value of m is computed, the number of solutions influencing the position update rule of other solutions will decrease over the course of iterations, until only the global best solution ($\vec{g}_t^{\,1}$) is used to influence the movement of the entire population. It is worth mentioning that, in (Mirjalili 2015a), the equation used to compute the value of m is given by $\mathsf{round}\big((n-t)\frac{(n-1)}{t_{max}}\big)$; however, this equation is wrong as it produces negative values when $t > n$.

The Moth-Flame Algorithm is PSO

The moth-flame algorithm is a variant of *e*PSO, where the only difference is that MFA uses a model of influence that assigns each particle with the personal best position of one specific neighbor depending on its ranking. Therefore, with the exception of the model of influence, the comparison between MFA and *e*PSO can be done directly, since it is possible to obtain the mathematical model of MFA (Equation 6.17) just by setting $\varphi_{1t} = 0$ and $k_1 = 1$ in the position update rule of *e*PSO (Equation 6.10).

In PSO, the model of influence (also known as *selection of social influence* (Mendes 2004) or *graph of influence* (Clerc 2010)) (MoI) refers to the way in which particles select other members of the swarm to influence their movement and it is equivalent to the concept of parental selection in evolutionary algorithms.

In Section 6.1, we describe the two most popular MoIs, which are the *best-of-neighborhood* and the *fully informed* models. However, there are many other models proposed in the literature of PSO, including *random*, in which particles are influenced by a random neighbor (Kennedy 1999), *ranked-fully informed* (Jordan et al. 2008), in which the contribution of each neighbor is weighted according to its rank—see (Mendes 2004; Montes de Oca 2011) for a detailed review.

Although, to the best of our knowledge, there is no PSO variant using exactly the same MoI implemented in the moth-flame algorithm, the idea of using a rank-based selection is not new at all in the metaheuristics literature. For example, the *ranking* selection mechanism used in EAs assigns to each individual a ranking based on its fitness value that determines the probability of selecting it for the next generation (Bäck et al. 1997; Grefenstette 2000). In the deterministic version of the ranking selection, the best individuals are selected from the population to pass to the next generation with the goal of both always preserving the best solutions found at any iteration and of biasing the creation of new solutions through recombination and mutation—see Section 6.2. In MFA, the best solution found by each particle is kept in its personal best vector ($\vec{p}_t^{\,i}$) and the only goal of using rankings is to create a mapping between a particle and one of its neighbors to bias its movement.

A PSO variant that is similar to MFA in this regard is the rank-based PSO with dynamic adaptation (PSO_{rank}) (Akbari and Ziarati 2011), where each particle receives influence from multiple neighbors and the contribution of each neighbor is weighted according to three criteria: ranking, Euclidean distance, and total number of neighbors. Similarly to MFA, in PSO_{rank}, the number of neighbors influencing the particles at each iteration is controlled using a parameter that decreases linearly according to the number of iterations, so that all particles are eventually influenced only by the global best solution.

The Metaphor of Moths Navigation

As we showed in the previous section, the moth-flame algorithm is the same as the *e*PSO algorithm except for the deterministic rank-based model of influence component, which is an idea originally proposed in the context of evolutionary algorithms and that is implemented in MFA in a similar way to PSO_{rank}. Therefore, the moth-flame algorithm does not meet with the criterion of novelty. In this section, we analyze the behavior of moths that inspired the algorithm to

check whether it has any component that can be useful from the point of view of designing an optimization algorithm.

The author of MFA says that the inspiration for this algorithm is the "navigation method of moths in nature" that allows them to move in a straight line by maintaining a fixed angle with respect to the moon—a mechanism known as *transverse orientation* according to Mirjalili (2015a). For developing MFA, the author considered the case when moths are attracted to artificial lights, not to the moon. In this case, moths engage in what the author referred to as "useless or dead spiral fly path", which happens because the transverse orientation method is only useful to fly in a straight line when the light source is very far. The flight of moths around artificial lights is modeled using Eqs. 6.17 and 6.18, where the set of current solutions ($\vec{x}_t$) represent "moths", their personal best positions ($\vec{p}_t$), called "flames", represent artificial light sources, and the "useless or dead spiral fly path" behavior is represented by computing the value of φ_t using a logarithmic spiral function.

Although the behavior that inspired MFA is moths' inability to escape from artificial light sources due to transverse orientation, in the algorithm, the author prevents this behavior by assigning "moths" (current solutions) to specific "flames" (personal best solutions) and by gradually stopping the use of the worst "flames" as the number of iteration grows. This is because, if "moths" and "flames" are defined as fixed couplings in the algorithm (as they are in the metaphor), a solution located in a poor quality region of the search space will most likely be incapable of moving away from that region because the only influence the solution has is its personal best solution. The author intended to avoid this problem by changing the specific personal best solution to which a current solution is assigned. However, this modification puts into question the motivation to use the metaphor of moths in the first place, since the algorithm following the "useless or dead spiral fly path" behavior of moths is, in the words of the author, prone "to be trapped in local optima quickly" (Mirjalili 2015a, p. 232).

After analyzing the "useless or deadly spiral fly path" behavior of moths that inspired MFA, the mathematical model derived from it, and the resulting algorithm that is "prone to stagnation quickly", it is obvious that the metaphor of "moths navigation" does not meet the criterion of usefulness. Also, considering the modification of assigning "moths" to specific "flames" that the author introduced to make the algorithm actually be able to perform optimization, radically changing the behavior of moths that inspired him to proposed MFA,

the use of this metaphor of "moths navigation" in the context of optimization is rather counterproductive.

6.3.3 Whale Optimization Algorithm

The whale optimization algorithm (WOA) (Mirjalili and Lewis 2016) is a combination of the mathematical models of GWO and MFA algorithms (all of them proposed by the same authors), in which solutions are updated using one of three possible position update rules (the three cases of Equation 6.19) that is chosen on the basis of stochastic criteria. The mathematical model of WOA is as follows:

$$\vec{x}_{t+1}^{\,i} = \begin{cases} \vec{x}_t^{\,k} - \varphi_{1t}(2\,\vec{r}_t - \vec{1}) \odot \left(2\,\vec{q}_t \odot \vec{x}_t^{\,k} - \vec{x}_t^{\,i}\right)^{\text{abs}} & \text{if } \texttt{Random\_Neighbor} \\ \vec{x}_t^{\,\text{best}} + \varphi_{2t}(\vec{x}_t^{\,\text{best}} - \vec{x}_t^{\,i})^{\text{abs}} & \text{if } \neg\texttt{Random\_Neighbor} \wedge \texttt{Exp\_Coefficient} \\ \vec{x}_t^{\,\text{best}} - \varphi_{1t}(2\,\vec{r}_t - \vec{1}) \odot \left(2\,\vec{q}_t \odot \vec{x}_t^{\,\text{best}} - \vec{x}_t^{\,i}\right)^{\text{abs}} & \text{if } \neg\texttt{Random\_Neighbor} \wedge \neg\texttt{Exp\_Coefficient} \end{cases} \tag{6.19}$$

where $(\cdot)^{abs}$ denotes the entrywise absolute value, $\vec{x}_t^{\,k}$ indicates the current position of a randomly chosen neighbor k at iteration t, and vectors $\vec{x}_t^{\,\text{best}}$, $\vec{r}_t$, $\vec{q}_t$, $\vec{x}_t^{\,i}$ and parameter φ_{1t} are the same ones defined for computing vector $\vec{s}_t^{\,1}$ in GWO (see Equation 6.14). In the remainder, we will refer to the three cases of Equation 6.19 as *first rule*, *second rule* and *third rule*, respectively, to simplify our analysis.

The computation of φ_{2t} in Equation 6.19 is given by

$$\varphi_{2t} = e^{\delta}\cos(2\,\pi\,\delta), \tag{6.20}$$

with $\delta = (\frac{-t}{t_{max}} - 2)\,\mathcal{U}[0,1] + 1$, that is exactly the same as Equation 6.18 proposed for MFA.

The specific position update rule that a particle will use depends on the value of the logical variables `Random_Neighbor` and `Exp_Coefficient`, which are defined as follows:

$$\begin{aligned} \texttt{Random\_Neighbor} &:= \begin{cases} \texttt{TRUE} & \text{if } \varphi_{1t}(2\,r_{1,t} - 1) > 1 \\ \texttt{FALSE} & \text{otherwise} \end{cases}, \\ \texttt{Exp\_Coefficient} &:= \begin{cases} \texttt{TRUE} & \text{if } \mathcal{U}[0,1] < 0.5 \\ \texttt{FALSE} & \text{otherwise} \end{cases}, \end{aligned} \tag{6.21}$$

where $r_{1,t}$ indicates the first element of vector $\vec{r}_t$. Because of the way the logical variables `Random_Neighbor` and `Exp_Coefficient` are used in Equation 6.19, the algorithm will select with higher probability the *first rule* during the first half of its execution, and either the *second rule* or the *third rule* (but not the *first* one) with the same probability during the second half. The reason why the *first rule* is not selected by the algorithm during the second half of its execution is that the value of φ_{1t} is lower than 1 and the probability of `Random_Neighbor:=TRUE` when $\varphi_{1t} < 1$ is 0.

The Whale Optimization Algorithm is PSO

WOA is a PSO algorithm that combines the mathematical models of StaPSO and *e*PSO. To explain how WOA compares with StaPSO, let us consider initially WOA's *first rule* and *third rule*. These two rules differ only in the vector that is used to bias the particles' movement: $\vec{x}_t^k$ (*first rule*) and $\vec{x}_t^{\text{best}}$ (*third rule*). In Section 6.3.1, where we compared GWO with PSO, we discussed the fact that the computation of vector $\vec{s}_t^1$ in GWO, which is the same as WOA's *third rule*, is defined in the same way as vectors $\vec{L}_t^i$ and $\vec{P}_t^i$ in StaPSO (Equation 6.5), with the only difference is that there is an additional perturbation component—$\varphi_{1t}(2\,\vec{r}_t - \vec{1})$—that was introduced to avoid the premature stagnation issue that affects GWO. Similarly, WOA's *first rule* is a variant of the mathematical model shown in Equation 6.5, but in this case there is also the difference that the local best particle ($\vec{l}_t^i$) is replaced by a randomly chosen neighbor ($\vec{x}_t^k$).

Unlike most PSO variants, WOA does not make use of a velocity vector ($\vec{v}_t^i$), which is an algorithm component that, among others, allows particles to diverge from moving exactly towards $\vec{l}_t^i$ or $\vec{p}_t^i$ and to explore other areas of the search space. To compensate for the lack of a component such as $\vec{v}_t^i$, WOA lets particles to be occasionally biased by $\vec{x}_t^k$ (*first rule*) instead of by $\vec{x}_t^{\text{best}}$ (*second* and *third rules*). Therefore, although WOA's *first rule* is defined in the same way as vectors $\vec{L}_t^i$ and $\vec{P}_t^i$ in StaPSO, the purpose of having this rule in the algorithm is rather similar to the one of using vector $\vec{v}_t^i$ in PSO algorithms—as it allows to introduce diversity in the solutions. To control the impact of the *first rule* in the optimization process and let particles converge towards $\vec{x}_t^{\text{best}}$, the authors of WOA defined the value of the logical variable `Random_Neighbor` as a function of the linearly decreasing parameter φ_{1t}, which results in `Random_Neighbor:=FALSE` in the second half of the algorithm's execution when $\varphi_{1t} < 1$.

Let us now consider the *second rule*—i.e., $\vec{x}_t^{\text{best}} + \varphi_{2t}(\vec{x}_t^{\text{best}} - \vec{x}_t^i)^{\text{abs}}$—that

allows particles to explore the area of the search space around the iteration-best solution and that uses the exponential function to compute the value of φ_{2t}. The ideas involved in the *second rule* of WOA are the same ideas introduced in *e*PSO (Equation 6.10) described in Section 6.1, where particles do not make use of vectors $\vec{p}_t$ and the value of the acceleration coefficients is computed using the exponential function. In fact, it is easy to see that the *second rule* of WOA can be obtained from the position update rule of *e*PSO (Equation 6.10) just by setting $\varphi_{1t} = 0$ and $k_1 = 1$, where the only difference is that, in WOA, the displacement of a particle towards $\vec{x}_t^{\text{best}}$ is adjusted using a random value (see Equation 6.18), whereas in *e*PSO it is adjusted based on the difference between $f(\vec{x}_t^i)$ and $f(\vec{g}_t)$.

The Metaphor of Humpback Whales' Bubble-Net

The authors of WOA say that the inspiration for this algorithm is the "bubble-net strategy" that humpback whales use for hunting (Mirjalili and Lewis 2016). According to the description they provided in their paper, this strategy involves performing two different *maneuvers*. The first maneuver is called "upward-spirals" and consists in creating an upward spiral path of bubbles in the water, while the second maneuver, called "double-loops", is composed of three different stages: *coral loop*, *lobtail* and *capture loop*. For developing WOA, the authors only considered the "upward-spirals" maneuver. The "double-loops" maneuver and its three stages are not described in the WOA paper.

The authors of WOA modeled the"spirals path of bubbles" created by whales during the "upward-spirals" maneuver by computing parameter φ_{2t} in WOA's *second rule* (Equation 6.19) using the exponential function. Since φ_{2t} is the only component in the algorithm that can be justified in terms of the metaphor of whales, the authors added two more maneuvers to the metaphor originally presented, one called "shrinking encircling" and the other called "search for prey". The "shrinking encircling" maneuver, whose mathematical model and description are exactly the same ones of the "encircling" step in the *grey wolf optimizer* (see Section 6.3.1) published by the same authors two years before WOA, is used to justify the linearly decreasing value of φ_{1t} in WOA's *first* and *third rules* and it is based on the idea that "whales can recognize the location of prey and encircle them"; whereas the "search for prey" maneuver is used to justify using the position vector of a randomly chosen neighbor k in WOA's *first rule* and it is based on the idea that "humpback whales search randomly according to the position of each other" (Mirjalili and Lewis 2016, pp. 53–54).

There are several reasons why the metaphor of "humpback whales hunting" that inspired WOA does not meet the criteria of usefulness and novelty. First, since the authors do not provide a complete description of the humpback whales' "bubble-net strategy" that inspired them, it is impossible to know what exactly is the optimization behavior they observed in it. Second, the only component taken from the behavior of humpback whales' "bubble-net strategy" presented in the WOA paper is the idea of computing the value of parameter φ_{2t} using the exponential function; however, as we discussed in the previous section, this idea was already proposed before in the PSO literature in a variant called *e*PSO. Third, the mathematical model of WOA is nothing but a combination of the ones used in the *grey wolf optimizer* (Section 6.3.1) and *moth-flame algorithm* (Section 6.3.2) algorithms, that were proposed by the same authors in 2014 and 2015, respectively, and that are variants of PSO.

Indeed, concerning the third point, in the WOA paper the authors mention as follows: "The main difference between the current work [*whale optimization algorithm*] and the recently published papers by the authors (particularly GWO (Mirjalili et al. 2014)) is the simulated hunting behavior with random or the best search agent to chase the prey and the use of a spiral to simulate bubble-net attacking mechanism of humpback whales" (Mirjalili and Lewis 2016, pp. 52). Although the authors acknowledge that there are similarities between WOA and GWO, they only make a vague mention of the connection between the two algorithms and fail to mention that the equation that models the spiral in the whales' bubble-net mechanism is the exact same equation they used to model the "useless or deadly spiral fly path of moths" in the moth-flame algorithm (Mirjalili 2015a).

6.3.4 Firefly Algorithm

In the firefly algorithm (FA) (Yang 2009), at each iteration t, each solution i in the population updates its position in the search space by moving towards every other solution that has higher quality than its own. The process to update solutions in this algorithm is carried out in two steps. In the first step, the population is sorted bottom-up according to the solutions' quality, so that the first solution to be updated is the one with the worst quality and the last one to be updated is the one with the best quality. In the second step, following the bottom-up order established before, each solution i determines the set W_t^i of solutions with better quality than its own; sets its initial position $\vec{m}_{t,s_0}^i = \vec{x}_t^i$; and

applies the following two equations:

$$\vec{x}_{t+1}^{\,i} = \vec{m}_{t,s_{|W_t^i|}}^{\,i}, \tag{6.22}$$

$$\vec{m}_{t,s}^{\,i} = \vec{m}_{t,s-1}^{\,i} + \varphi_t^{\vec{w}_{t,s}^{\,i},\vec{m}_{t,s-1}^{\,i}}\left(\vec{w}_{t,s}^{\,i} - \vec{m}_{t,s-1}^{\,i}\right) + \xi\vec{r}_{t,s}^{\,i}, \tag{6.23}$$

where $\vec{w}_{t,s}^{\,i}$ is an element of the ordered set W_t^i, $\varphi_t^{\vec{w}_{t,s}^{\,i},\vec{m}_{t,s-1}^{\,i}}$ is an acceleration coefficient whose value is computed as a function of the Euclidean distance between the two intermediate points $\vec{w}_{t,s}^{\,i}$ and $\vec{m}_{t,s-1}^{\,i}$, $\vec{r}_{t,s}^{\,i}$ is a vector whose components are random numbers drawn from the uniform distribution $\mathcal{U}[0,1]$, and ξ is a parameter. (In the remainder of this analysis, we will use the shorter notation $\varphi_t^{\vec{w},\vec{m}}$ as the meaning is clear from the context.) As it can be observed by the way Equation 6.23 is defined, a solution updates its position by performing $|W_t^i|$ movements, one for each solutions in W_t^i, where the position obtained in movement $s-1$ (indicated by $\vec{m}_{t,s-1}^{\,i}$) is the starting position for the next one ($\vec{m}_{t,s}^{\,i}$). Also, it is worth noticing that, since the best quality solution has an empty set W_t^i, the new position of this solution is obtained by adding a random vector $\xi\vec{r}_{t,s}^{\,i}$ to its current position.

The acceleration coefficient $\varphi^{\vec{w},\vec{m}}$ is computed as follows:

$$\varphi^{\vec{w},\vec{m}} = \iota \cdot e^{-\gamma|\vec{w}-\vec{m}|^2}, \tag{6.24}$$

where $|\vec{w}-\vec{m}|$ is the Euclidean distance between solutions $\vec{w}$ and $\vec{m}$, and γ and ι are two parameters that allow to control, respectively, the weight given to $|\vec{w}-\vec{m}|^2$ and to the exponential function. Because of the way in which $\varphi^{\vec{w},\vec{m}}$ is defined, solutions have larger displacements when they are located close to each other and smaller ones when they are located far away.

The Firefly Algorithm is PSO

FA is a variant of the SDPSs proposed by Peña (2008a,b) using the extrapolation coefficients of *e*PSO and the fully informed model of FiPSO. In order to explain why FA is a combination of these PSO algorithms, we will consider two cases: $|W_t^i| = 1$ and $|W_t^i| > 1$.

In the first case (i.e., when $|W_t^i| = 1$), a particle updates its position in only one movement, which allows to combine Eqs. 6.22 and 6.23 into one single

equation as follows:

$$\vec{x}_{t+1}^{\,i} = \vec{x}_t^{\,i} + \varphi_t^{i,\vec{w}_t^{\,i}}\left(\vec{w}_t^{\,i} - \vec{x}_t^{\,i}\right) + \zeta\vec{r}_t^{\,i}. \tag{6.25}$$

As it can be easily seen, it is possible to obtain Equation 6.25 from the position update rule of SDPSs (Equation 6.7) by setting $\vec{y} = \vec{w}_t^{\,i}$, $\epsilon = \varphi_t^{i,\vec{w}_t^{\,i}}$, and by computing the value of $\varphi_t^{i,\vec{w}_t^{\,i}}$ using the strategy proposed in *e*PSO (Equation 6.10). The only difference is that, in FA (Equation 6.24), the value of $\varphi_t^{i,\vec{w}_t^{\,i}}$ is adjusted in terms of particles' Euclidean distance, while in *e*PSO (Equation 6.10) it is adjusted in terms of the difference between their objective function evaluation. Except for this minor difference, when $|W_t^i| = 1$, the position update rule of FA is a special case of the one used in SDPSs with extrapolation coefficients (*e*PSO).

In the second case (i.e., when $|W_t^i| > 1$), the mathematical model of FA (that is, Eqs. 6.22 and 6.23) involves the recursive addition of $|W_t^i|$ exponentially weighted vectors. In this case, the mathematical model of FA is the result of using the same algorithm components mentioned before (i.e., the position update rule of SDPSs (Equation 6.7) and the extrapolation coefficients of *e*PSO (Equation 6.10)) with a particular case of the fully informed model of FiPSO (Equation 6.6). Differently from FiPSO, where particles add as many vectors as their number of neighbors, in the version of the fully informed model used in FA, particles only add the vectors of those neighbors with better quality than their own.

The Metaphor of Fireflies Flashing/Brightening

Although the author of FA says that the algorithm is inspired by the "flashing behavior of fireflies", which consists of *short*, *rhythmic flashes* that fireflies produce (Yang 2009, p.171), the only idea the author used to develop the algorithm is that "fireflies are attracted towards other brighter fireflies".

Most of the metaphor of "fireflies brightening" (as opposed to the one of "fireflies flashing") is explained in terms of the different set of values that can be obtained by varying the value of parameter γ (Equation 6.24), for which the author considered two limit cases. The first case is when γ goes to 0 and the value of $\varphi^{\vec{w},\vec{m}}$ goes to 1, making the attraction among fireflies constant regardless of their distance in the search space. In the imagery of the fireflies metaphor, this is the case when "the light intensity does not decay in an idealized sky" and "fireflies can be seen anywhere in the domain" (Yang 2009, p. 174). The

second case is when γ goes to ∞ and the value of $\varphi^{\vec{w},\vec{m}}$ goes to 0, which makes the attractiveness among fireflies negligible and new solutions are created only by means of the random perturbation $\tilde{\varsigma}\vec{r}^{\,i}_{t,s}$ (see Equation 6.23). According to the author, this is the case when fireflies are either "short-sighted because they are randomly moving in a very foggy region", or (for reasons not explained in the paper) "fireflies feel almost zero attraction to other fireflies."

As we mention before, the firefly algorithm is not really inspired by the behavior of "fireflies flashing", but on the phenomenon of light attenuation (i.e., the reduction in intensity of a light beam as the beam propagates in matter due to the joint action of the absorption and scattering of light) and the connection that the author makes between light intensity and the objective function of an optimization problem. For the author of FA, since fireflies produce light through bio-luminescence, they can represent candidate solutions for an optimization problem, and since the quality of the solution can be associated with the intensity of the light it emits, it follows that: the "brighter" the "firefly", the better the solution it represent and the more "attractive" it becomes to other "fireflies".

The reasons why FA does not meet the criterion of usefulness are: (i) it is unclear what is the optimization behavior that the author observed in the behavior of "fireflies flashing" that can be used to develop a new optimization algorithm; (ii) the central elements in the "fireflies flashing" behavior (i.e., *short* and *rhythmic* light flashes) are not considered in the algorithm design of FA; and (iii) in the metaphor of "fireflies brightening", the association between "fireflies" and "brightness" only adds a new, unnecessary terminology to refer to the concepts of *solution* and *solution quality*. In addition to this, FA does not fulfill the criterion of novelty because, as we showed in the previous section, FA uses the same ideas that were proposed before in SDPSs, *e*PSO and FiPSO.

6.3.5 Bat Algorithm

The bat algorithm (BA) (Yang 2010) is a population-based algorithm in which new solutions are generated in two possible ways: (i) by identifying good search directions that are estimated based on the position of $\vec{g}_t$, or (ii) by generating a random point around $\vec{g}_t$ and accepting it on the basis of stochastic criteria. To do so, each solution has two parameters associated: ρ^i_t, which is the probability of randomly generating a solution around $\vec{g}_t$ that increases over time, and ζ^i_t, which is the probability of accepting the new solution that decreases over time.

This position update rule of BA is the following:

$$\vec{x}_{t+1}^{i} = \begin{cases} \vec{g}_t + \hat{\zeta}_t \vec{r}_t^{i}, & \text{if } \texttt{Generate} \wedge \texttt{Accept} \\ \vec{x}_t^{i} + \vec{v}_{t+1}^{i} & \text{if } (\texttt{Generate} \wedge (\neg\texttt{Accept})) \vee (\neg\texttt{Generate}) \end{cases}, \tag{6.26}$$

where $\hat{\zeta}_t$ is the average of the parameters ζ_t^i of all the solutions in the population, and $\vec{r}_t^{i}$ is a vector with values randomly distributed in $\mathcal{U}[-1,1]$. The logical variables `Generate` and `Accept` are defined as follows:

$$\begin{aligned} \texttt{Generate} &:= \begin{cases} \texttt{TRUE} & \text{if } \rho_t^i > \mathcal{U}[0,1] \\ \texttt{FALSE} & \text{otherwise} \end{cases}, \\ \texttt{Accept} &:= \begin{cases} \texttt{TRUE} & \text{if } \left(f(\vec{z}_t^{i}) < f(\vec{g}_t)\right) \wedge (\mathcal{U}[0,1] < \zeta_t^i) \\ \texttt{FALSE} & \text{otherwise} \end{cases}, \end{aligned} \tag{6.27}$$

where $f(\cdot)$ refers to the objective function of a minimization problem.

In BA, at each iteration t and with probability ρ_t^i, a solution i generates a random point around $\vec{g}_t$ that it keeps in a variable $\vec{z}_t^{i}$. The newly generated point $\vec{z}_t^{i}$ is accepted as the new position of i only when `Accept`:=`TRUE`, which happens when two conditions are met: first, the quality of $\vec{z}_t^{i}$ is higher than that of $\vec{g}_t$, and second, $\vec{z}_t^{i}$ is accepted with probability ζ_t^i.

In the case when either `Generate`:=`FALSE` (i.e., the random solutions was never generated) or `Accept`:=`FALSE` (i.e., $\vec{z}_t^{i}$ was rejected), solution i generates a velocity vector ($\vec{v}_t^{i}$) that is added to its current position $\vec{x}_t^{i}$, as shown in the second case of Equation 6.26. Vector $\vec{v}_{t+1}^{i}$ is computed as follows:

$$\vec{v}_{t+1}^{i} = \vec{v}_t^{i} + \vec{d}_t^{i} \odot (\vec{g}_t - \vec{x}_t^{i}) \tag{6.28}$$

with

$$\vec{d}_t^{i} = \varphi_{min} + \vec{a}_t^{i}(\varphi_{max} - \varphi_{min}), \tag{6.29}$$

where $\varphi_{min} < \varphi_{max}$ are two parameters and $\vec{a}_t^{i}$ is a vector whose values are sampled from $\mathcal{U}[0,1]$.

The equations to update the probabilities ρ_t^i and ζ_t^i are:

$$\rho_{t+1}^i = \rho_0(1 - e^{-\beta_1 t'})$$
$$\zeta_{t+1}^i = \begin{cases} \beta_2\, \zeta_t^i & \text{if } \texttt{Generate} \wedge \texttt{Accept} \\ \zeta_t^i & \text{otherwise} \end{cases} \tag{6.30}$$

where $\beta_1 > 0$ and $0 < \beta_2 < 1$ are parameters, t' is an iteration counter that is updated every time $\texttt{Generate} \wedge \texttt{Accept} := \texttt{TRUE}$, and ρ_0 is the initial value of parameter ρ. Note that, in Equation 6.30, the value of ρ_t^i tends to ρ_0 and the value of ζ_t^i tends to 0. Also, note that, as the value ζ_t^i decreases with the number of iterations, so does the value of $\hat{\zeta}_t$; therefore, for increasing t values, the solutions generated in the first case of Equation 6.26 will be closer and closer to $\vec{g}_t$.

The Bat Algorithm is PSO

The bat algorithm is a simplified variant of the standard PSO algorithm (StdPSO) combined with a simulated annealing (SA) acceptance criterion (Černý 1985; Kirkpatrick et al. 1983). First, in order to show that BA is a simplified variant of StdPSO, we compare Eqs. 6.26, 6.28 and 6.29 of BA with the position (Equation 6.1) and velocity (Equation 6.2) update rules of StdPSO.

In BA, the second case of Equation 6.26 is exactly the same as the position update rules in StdPSO (Equation 6.1), that consist in adding a velocity vector to a particle's current position. Also, the velocity vector is computed in the same way in both algorithms. By setting $\omega = 1$ and $\varphi_1 = 0$, the velocity update rule of StdPSO (Equation 6.2) simplifies to the one of BA (Equation 6.28). The perturbation component $\vec{d}_t^i$ in BA (Equation 6.29) is equivalent to the term $\varphi_2 \vec{b}_t^i$ in StdPSO (Equation 6.2). The only difference is that the value of the acceleration coefficient φ in Equation 6.29 is computed in the range $[\varphi_{min}, \varphi_{max}]$ with the goal of varying the magnitude of the perturbation induced by $\vec{a}_t^i$. One of the first PSO variant using the idea of varying the value of the control parameters of PSO is the "time-varying acceleration coefficient PSO" (Ratnaweera et al. 2004), in which the value of φ_2 linearly increases from φ_{min} to φ_{max}.

Now, we will compare BA with simulated annealing (SA) (Kirkpatrick et al. 1983), which is a single solution based algorithm for solving combinatorial optimization problems proposed in the early 80's. As shown in Equation 6.29, BA uses the concept of generating new solutions around $\vec{g}_t$ and accepting

them on the basis of a decreasing probability (parameter ζ_t^i). This idea comes originally from SA, where a parameter called *temperature* (T) that decreases over time is used to decided when to accept a worsening solution. In the context of SA, the so-called *cooling scheme* is used to control the updates of parameter T iteration after iteration (Franzin and Stützle 2019). In BA, parameter ζ_t^i plays the same role as parameter T in SA. Also, the model used in BA to update the value of ζ_t^i (Equation 6.30) is the same as the well-known *geometric cooling scheme*, which was proposed in the very first version of SA by Kirkpatrick et al. (1983). One minor difference is that, in BA, the value of ζ_t^i is updated only when a solution is accepted, while in SA the value of T is typically updated at the end of each iteration.

The Metaphor of Bats Echolocation

The author of BA says that the inspiration for this algorithm is "the behavior of echolocation that some bats species use to find preys, avoid obstacles and discriminate between different objects" (Yang 2010, p.66). For developing BA, the author "idealized" several aspects of this behavior. In the words of the author, it was assumed that: (i) "all bats use echolocation for sensing distance", (ii) "bats are able to differentiate in some magical way between food/prey and background barriers", (iii) "bats can automatically adjust the frequency and rate in which they are emitting sound", and (iv) "the loudness of their sound can only decrease from a large value to a minimum constant" (Yang 2010, pp. 67–68).

To explain BA, the author considered that bats have two different flying modes, which correspond to the two cases in Equation 6.26. In the first mode, bats fly randomly adjusting their "pulse emission rate" ρ_t^i and "loudness" ζ_t^i. According to the author, small values for parameter ρ_t^i and large ones for ζ_t^i represent when "bats are randomly searching for a prey", while the opposite case (i.e., large values for ρ_t^i and small ones for ζ_t^i) represent when "bats have found a prey and temporarily stop emitting any sound" (Yang 2010, p.70). The mathematical model of bats adjusting their "pulse emission rate" and "loudness" is given by Equation 6.30. For the second flying mode, which is modeled using Eqs. 6.28 and 6.29, the authors considered that bats control their step size and range of movement by adjusting their "sound frequency" (modeled by vector $\vec{d}_t^i$ in Equation 6.29) and by moving towards the best bat in the swarm; parameters φ_{min} and φ_{max} represent "the range of frequencies in which bats emit their

sound".

As there are so many simplifications and unrealistic assumptions in the behavior of "bats echolocation" as idealized by Yang (2010), it is impossible to understand how this behavior was taken into account at all for developing BA. Consider, for example, the unlikely ideas that bats use the location of the global best bat while hunting, or that the loudness of their sound can only decrease, or that they have a "magical" ability to differentiate between food/prey and background barriers. For this reason, BA does not meet the criterion of usefulness. Also, as we showed in the previous section, BA uses concepts originally proposed in StdPSO and SA that were published, respectively, in 1995 and 1983, thereby failing also in the criterion of novelty. Unfortunately, it seems evident to us that the only goal of using the metaphor of "bats echolocation" has been to hide the fact that the algorithm is unnecessary as its only contribution is a bats-inspired terminology to refer to well-known concepts.

6.3.6 Antlion Optimizer

In the antlion optimizer (ALO) (Mirjalili 2015b), a population of μ solutions are randomly selected for recombination with the global best solution ($\vec{g}_t$) in order to produce λ new solutions, with $\lambda = \mu$. In ALO, at the beginning of each iteration, each solution in the population P is given a probability of being selected for recombination with $\vec{g}_t$ as follows:

$$\Pr(\vec{x}_t^k) = \frac{\texttt{fit}(\vec{x}_t^k)}{\sum_{z=1}^{|P|} \texttt{fit}(\vec{x}_t^z)}, \tag{6.31}$$

where $\Pr(\vec{x}_t^k)$ is the probability of selecting solution $\vec{x}_t^k \in P$ and $\texttt{fit}(\cdot)$ is a function that maps the quality of a solution with a real positive value, so that the better the quality of a solution, the greater the value returned by $\texttt{fit}(\cdot)$. It is worth noticing that, since $\vec{g}_t$ is an element of P, the solution with higher probability of being selected for recombination with $\vec{g}_t$ is the solution $\vec{g}_t$ itself.

The equation used to recombine solutions in order to create the set of λ new solutions is:

$$\vec{x}_{t+1}^i = \frac{\vec{x}_t'^k + \vec{g}_t'^i}{2} \text{ for } i = 1, \ldots, \lambda \tag{6.32}$$

where $\vec{x}_t'^k$ is a vector generated by perturbing solution $\vec{x}_t^k$ that has been randomly selected with repetitions from P based on the probabilities computed

using Equation 6.31, and $\vec{g}_t'^i$ is a vector generated by perturbing $\vec{g}_t$.

The computation of vectors $\vec{x}_t'^k$ and $\vec{g}_t'^i$ is done using an elaborated procedure that involves performing a random walk of s-steps for each dimension of each vector. This procedure is defined as follows:

$$u^j = \frac{\left(\mathcal{R}^j_{t,s=\varrho} - \texttt{min}(\mathcal{R}^j_t)\right) \times (\alpha^j_t - \eta^j_t)}{\texttt{max}(\mathcal{R}^j_t) - \texttt{min}(\mathcal{R}^j_t)} + \eta^j_t, \text{ for } j = 1, \ldots, d; \text{ for } \vec{u} \in \{\vec{x}^k_t, \vec{g}_t\} \tag{6.33}$$

where vector $\vec{u}$ is a variable used to iterate between vectors $\vec{x}^k_t$ and $\vec{g}_t$, $\mathcal{R}^j_t$ is a sequence of ϱ values computed using a 1-dimensional random walk on $\mathbb{Z}$ that starts at 0 and moves $+1$ or -1 with equal probability at each step s, $\mathcal{R}^j_{t,s=\varrho}$ indicates the last value in the sequence $\mathcal{R}^j_t$, and functions $\texttt{min}(\cdot)$ and $\texttt{max}(\cdot)$ return, respectively, the minimum and maximum values in $\mathcal{R}^j_t$. Vectors $\vec{\alpha}_t$ and $\vec{\eta}_t$, which are computed as:

$$\vec{\alpha}_t = \begin{cases} (x^j_u/\psi_t) + u^j & \text{if } \mathcal{U}(0,1) > 0.5 \\ (-x^j_u/\psi_t) + u^j & \text{otherwise} \end{cases}, \forall j \tag{6.34}$$

and

$$\vec{\eta}_t = \begin{cases} (x^j_l/\psi_t) + u^j & \text{if } \mathcal{U}(0,1) > 0.5 \\ (-x^j_l/\psi_t) + u^j & \text{otherwise} \end{cases}, \forall j, \tag{6.35}$$

allow to use a fraction $1/\psi_t$ of the upper (x^j_u) and lower (x^j_l) bounds of each dimension of the vector being perturbed to normalize the values of the random walk $\mathcal{R}^j_t$ around them. Finally, to create the population that will be used in the next iteration, the best μ solutions are selected from the set of $\mu + \lambda$ solutions at the end of each iteration.

The Antlion Optimizer is an ES

As the reader familiar with evolutionary algorithms must have realized by now, the algorithm components proposed for ALO are the same as those proposed in the context of evolution strategies (ESs)—see Section 6.2. In particular, Equation 6.31 corresponds to the *fitness proportional* parental selection, Equation 6.32 corresponds to the *intermediate* recombination, and Equation 6.33 corresponds to the *spherical/isotropic* mutation. In fact, as ALO uses the *elitist* $(\mu + \lambda)$ survival selection mechanism, it is virtually the same as the $(\mu + \lambda)$–ES (Schwefel

1981), where the only difference is that, in ALO, mutation is applied before recombination, while in the $(\mu + \lambda)$–ES it is applied after recombination.

Although most of the components in ALO can be easily recognized as evolutionary operators, this may not be necessarily the case for the *spherical/isotropic* mutation. The mutation operator implemented in ALO (Equation 6.33) uses a complex procedure that involves computing a random walk (Equation 6.33) and two other equations (Eqs. 6.34 and 6.35) to mutate each dimension of a vector, whereas the procedure implemented in ESs (Equation 2.1) requires sampling normally distributed values. The 1-dimensional random walk over $\mathbb{Z}$ implemented in ALO (that starts at 0 and adds $+1$ or -1 with equal probability at each time step s) is commonly known in the literature as the *simple isotropic random walk* (Codling et al. 2008) and its diffusion model is given by the Gaussian distribution with mean 0 and variance s for a sufficiently large number of time steps s. Using this fact, the complex procedure proposed for ALO in Equation 6.33 to compute vectors $\vec{x}_t'^k$ and $\vec{g}_t'^i$ can be reformulated in a much simpler and computationally efficient way as follows:

$$\vec{x}_t'^k = \vec{x}_t^k + \mathcal{N}\big(0, \psi_t \mathbf{I}\big), \tag{6.36}$$

$$\vec{g}_t'^i = \vec{g}_t + \mathcal{N}\big(0, \psi_t \mathbf{I}\big), \tag{6.37}$$

where ψ_t is the time-varying parameter that the authors of ALO introduced in Eqs. 6.34 and 6.35.

The Metaphor of Antlions Hunting

The author of ALO says that the inspiration for the algorithm is the "intelligent behavior of antlions in hunting ants in nature" (Mirjalili 2015b, p.81). According to the description of this behavior provided in the ALO paper, the strategy that antlions use for hunting consists in the following steps: *(i) digging a cone-shaped pit in the sand, (ii) hiding underneath the sand at the bottom of the pit, and (iii) waiting for prey to fall in the pit so that they can catch it*.

In ALO, the μ solutions at the beginning of each iteration represent "antlions" and the fitness value of the solutions—computed using $\texttt{fit}(\cdot)$ in Equation 6.31—represent "the size of the pit in which the antlion constructed and is hidden at the bottom". Accordingly, in the imagery of the metaphor, the "fitter" the "antlion", the larger its pit and the higher its chances of catching an ant. To model the behavior of ants randomly moving in the sand, the author used Eqs. 6.33,

6.34 and 6.35 that involve performing a random walk on $\mathbb{Z}$. For Mirjalili (2015b), the computation of vectors $\vec{x}_t'^k$ and $\vec{g}_t'^i$ in Equation 6.33 represent the "influence of antlions pits on the random walk of ants", and computing the arithmetic average of those two vectors (Equation 6.32) represents "the final position of the ant". Also, when a new solution has higher quality than the one used to create it—i.e., when $f(\vec{x}_{t+1}^i)$ is better than $f(\vec{x}_t'^k)$—this represents the case when "an ant reaches the bottom of the pit and is caught in the antlion's jaw" (Mirjalili 2015b, p.83). Finally, the fact that vector $\vec{g}_t$ is always used when recombining two solutions (Equation 6.32) represents the fact that "each ant can be caught by an antlion in each iteration and the elite (fittest antlion)" (sic) (Mirjalili 2015b, p.82).

As the reader should have realized by now, the metaphor of "antlions hunting" is nothing but a far-fetched way to explain the proposed antlion optimizer. The "novel" antlion optimizer proposed in 2015 does not meet the criterion of novelty because, as we showed in the previous section, it is a variant of the $(\mu + \lambda)$–ES proposed roughly 35 years before ALO. Also, the mathematical model derived from the metaphor of "antlions hunting" resulted in an inefficient procedure to perturb solutions (Eqs. 6.33, 6.34 and 6.35) that produces the same kind of perturbation as the simple *spherical/isotropic* mutation (Equation 2.1). Therefore, ALO does not meet the criterion of usefulness. Finally, it is worth pointing out that, similarly to what we observed for the other metaphors analyzed in this chapter, many of them proposed by the same author, it is impossible to understand what is the optimization process/component observed by the author in the behavior of antlions that inspired him to propose this algorithm.

6.3.7 Cuckoo Search

While carrying out our analysis of cuckoo search, we found that the algorithm proposed in (Yang and Deb 2009, 2010) and the implementation provided by the authors in Matlab in (Yang 2021) are quite different; moreover, neither of them really follows the metaphor of the cuckoos that inspired the algorithm. In the remainder of this section, similarly to how we did for the other six algorithms, we describe cuckoo search in plain computational terms, compare it with known approaches, and analyze the metaphor that inspired the algorithm; however, we present all this information in a slightly different way. After presenting the implementation of the algorithm, we examine the metaphor of the "cuckoos

parasitic behavior" and analyze the way it was (mis)used to develop the cuckoo search algorithm. Then, we highlight the many inconsistencies that exist among the metaphor, the algorithm and the implementation of cuckoo search. Lastly, we show that the implementation of cuckoo search in Matlab (which is referred by its authors as the correct way to implement the algorithm) is equivalent to the $(\mu + \lambda)$–ES using the mutation and recombination operators proposed in DE.

The Implementation of Cuckoo Search

According to the publicly available implementation in Matlab (Yang 2021), cuckoo search is an iterative, population-based algorithm that consists of the following four steps.

Step 1 *(initialization)*. Create a set of μ initial solutions $\vec{x}_t^{\,i}$ randomly distributed in the search space using the following equation:

$$x_{t=0}^{i,k} = \mathcal{U}(lb^k, ub^k), \text{ for } i = 1, \ldots, \mu \text{ and } k = 1, \ldots, d, \tag{6.38}$$

where t is the iteration number, $\mathcal{U}$ is a random uniform distribution, lb^k and ub^k are the lower and upper limit of dimension k, and d is the number of dimensions in the problem.

Step 2 *(perturbation)*. Perturb all μ solution $\vec{x}_t^{\,i}$ by adding a random vector $\vec{r}_t^{\,i}$ as follows:

$$\vec{x}_t^{\,i'} = \vec{x}_t^{\,i} + \alpha\, \vec{r}_t^{\,i}, \text{ for } i = 1, \ldots, n, \tag{6.39}$$

where $\vec{x}_t^{\,i'}$ is the perturbed solution, $\vec{r}_t^{\,i}$ is a random vector whose components are sampled from a Lévy distribution $\mathcal{L}_\gamma$ with scale parameter γ, and α is a parameter that controls the magnitude of the perturbation.

Step 3 *(selection)*. Compare each pair $(\vec{x}_t^{\,i}, \vec{x}_t^{\,i'})$ on the basis of the objective function $f(\cdot)$ and select the one that has higher quality. This is formally done as follows:

$$\vec{x}_{t'}^{\,i} = \begin{cases} \vec{x}_t^{\,i'}, & \text{if } f(\vec{x}_t^{\,i'}) \text{ is better than } f(\vec{x}_t^{\,i}) \\ \vec{x}_t^{\,i}, & \text{otherwise} \end{cases}. \tag{6.40}$$

Step 4 *(recombination)*. With probability $1 - p_a$, apply recombination to the k^{th} component of vector $\vec{x}_{t'}^{\,i}$ using two randomly selected solutions $\vec{x}_{t'}^{\,l^i} \in L_t$ and $\vec{x}_{t'}^{\,m^i} \in M_t$, where sets L_t and M_t contain each a copy of the population

after executing step 3 (selection), i.e., a copy of $\vec{x}_{t'}^{\,i}$ for $i = 1, \ldots, n$. Step 4 *(recombination)* is computed as follows:

$$x_{t+1}^{i,k} = \begin{cases} x_{t'}^{i,k} + \mathcal{U}[0,1] \cdot (x_{t'}^{l^i,k} - x_{t'}^{m^i,k}), & \text{if } \mathcal{U}[0,1] \geq p_a \\ x_{t'}^{i,k}, & \text{otherwise} \end{cases}, \forall k, \forall i. \quad (6.41)$$

Solutions $\vec{x}_{t'}^{\,l^i}$ and $\vec{x}_{t'}^{\,m^i}$ are selected from sets L_t or M_t without replacement, that is, each solution is used once as $\vec{x}_{t'}^{\,l}$ and once as $\vec{x}_{t'}^{\,m}$. Note that it can be the case that $\vec{x}_t^{\,l^i} = \vec{x}_t^{\,m^i} = \vec{x}_t^{\,i}$, in which case vector $\vec{x}_t^{\,i}$ is not modified. After finishing the process of recombination, solutions are evaluated once again.

The implementation of cuckoo search consists in applying step 1 *(initialization)* once and then repeating step 2 *(perturbation)*, step 3 *(selection)* and step 4 *(recombination)* iteratively until a termination criterion is met.

The Metaphor of Cuckoos' Parasitic Breeding Behavior

In the first two articles proposing cuckoo search (Yang and Deb 2009, 2010), which are the ones typically cited to reference the algorithm,[3] the authors describe the cuckoo search algorithm using as a metaphor the "parasitic breeding behavior of cuckoos", a behavior that, according to them and to the reference that they cite in their article, some species of cuckoos practice. In the words of the authors:

> *"Cuckoos are fascinating birds, not only because of the beautiful sounds they can make, but also because of their aggressive reproduction strategy. Some species such as the ani and guira cuckoos lay their eggs in communal nests, though they may remove others' eggs to increase the hatching probability of their own eggs (Payne et al., 2005). Quite a number of species engage the obligate brood parasitism by laying their eggs in the nests of other host birds (often other species). There are three basic types of brood parasitism: intraspecific brood parasitism, cooperative breeding and nest takeover. Some host birds can engage direct conflict with the intruding cuckoos. If a host bird discovers the eggs are not its own, it will either throw these alien eggs away or simply abandons its nest and builds a new nest elsewhere. Some cuckoo species such as the new world brood-parasitic Tapera have evolved in such a way that female parasitic cuckoos are often very specialised in*

[3] (Yang and Deb 2009): 7,729 citations; and (Yang and Deb 2010): 2,843 citations. Source: Google Scholar. Retrieved: June 12, 2023.

> *the mimicry in colour and pattern of the eggs of a few chosen host species (Payne et al., 2005). This reduces the probability of their eggs being abandoned and thus increases their reproductivity."*
> *(Yang and Deb 2009, p. 210) and (Yang and Deb 2010, pp. 331, 332)*

> *"The timing of egg-laying of some species is also amazing. Parasitic cuckoos often choose a nest where the host bird just laid its own eggs. In general, the cuckoo eggs hatch slightly earlier than their host eggs. Once the first cuckoo chick is hatched, the first instinct action it will take is to evict the host eggs by blindly propelling the eggs out of the nest, which increases the cuckoo chick's share of food provided by its host bird (Payne et al., 2005). Studies also show that a cuckoo chick can also mimic the call of host chicks to gain access to more feeding opportunity."*
> *(Yang and Deb 2009, p. 210) and (Yang and Deb 2010, p. 332)*

To translate the metaphor described above into an algorithm, the authors simplified the process into three idealized rules. In the words of the authors:

> *"For simplicity in describing our new Cuckoo Search (Yang and Deb 2009), we now use the following three idealized rules:*
> *– Each cuckoo lays one egg at a time, and dumps it in a randomly chosen nest;*
> *– The best nests with high quality of eggs (solutions) will carry over to the next generations;*
> *– The number of available host nests is fixed, and a host can discover an alien egg with a probability $p_a \in [0, 1]$. In this case, the host bird can either throw the egg away or abandon the nest so as to build a completely new nest in a new location. For simplicity, this last assumption can be approximated by a fraction p_a of the n nests being replaced by new nests (with new random solutions at new locations)."*
> *(Yang and Deb 2010, p. 332)*

In addition to these rules, the description of cuckoo search is limited to one equation that is used to generate new solutions as follows:

> *"When generating new solutions $\vec{x}^{(t+1)}$ for, say, a cuckoo i, a Lévy flight is performed.*
>
> $$\vec{x}_i^{(t+1)} = \vec{x}_i^t + \alpha \otimes \text{Lévy}(\lambda) \tag{6.42}$$
>
> *" (Yang and Deb 2009, p. 211) and (Yang and Deb 2010, p. 333).*

The authors of cuckoo search refer to Equation 6.42 as "Lévy flights" because it makes use of the Lévy distribution to sample random numbers. Note that,

because of the use of the metaphor of cuckoos, the authors introduced new terminology, in particular, they used three different words (*eggs*, *nests* and *cuckoos*) to refer to a candidate solution to the problem—see the three idealized rules above.

However, this terminology is not clear and the authors are not consistent with its use. If we consider the first rule: "*Each cuckoo lays one egg at a time, and dumps it in a randomly chosen nest*" that is modeled using Equation 6.42, it is understood that $\vec{x}_i^t$ is the position of the *cuckoo*, the term $\alpha \otimes \text{Lévy}(\lambda)$ that is added represents the distance the *cuckoo* flew, and $\vec{x}_i^{(t+1)}$ is the nest to which the *cuckoo* arrived and deposited the *egg*. However, in (Yang and Deb 2009, p. 211), the terminology seems to be used differently:

> *"For simplicity, we can use the following simple representations that each egg in a nest represents a solution, and a cuckoo egg represent a new solution, the aim is to use the new and potentially better solutions (cuckoos) to replace a not-so-good solution in the nests."*

According to this "representation", what the authors refer to as an *egg* and a *cuckoo* is inverted with regard to their first rule and to Equation 6.42; that is, in the excerpt above, the *egg* represents the initial solution $\vec{x}_i^t$ and the *cuckoo* represents the new and potentially better solution $\vec{x}_i^{(t+1)}$.

The most serious problem with the metaphor, however, comes from the second rule, which says that "*The best nests with high quality of eggs (solutions) will carry over to the next generations*". While the metaphor of the *cuckoos' parasitic behavior* describes a process in which cuckoos lay their eggs in the nest of other birds and some of these eggs survive and some others do not (as specified in the third rule), there is no mention of a selection mechanism to get rid of the low quality eggs that were laid in the different nests. However, by including the second rule as part of the rules that define the cuckoo search algorithm and saying that these rules are taken from their cuckoos metaphor, the authors implied that selection is part of the cuckoos metaphor when it is not.

Inconsistencies Among the Components of Cuckoo Search

There are three important components that should help understanding how cuckoo search works: the metaphor, the algorithm description and its implementation. Unfortunately, we discovered that the concepts brought forward by the metaphor are hardly used in the algorithm and that the algorithmic procedure

proposed for *cuckoo search* in (Yang and Deb 2009, 2010) and the implementation provided by its authors in Matlab in (Yang 2021) are quite different.

Differences between metaphor and algorithm — The authors of cuckoo search say that they were inspired by the *cuckoos' parasitic reproduction* behavior, that is, by the cuckoos' strategy of laying eggs in the nest of other birds; and by the fact that cuckoos' eggs laid in the nests of other birds are sometimes identified by those other birds, that can either remove them from the nest or abandon them in the nest. They translated this behavior into a set of rules as follows:

i at each iteration, each cuckoo lays one egg in a randomly chosen nest;
ii the number of nests is fixed and each nest can host only one egg;
iii the best quality eggs at the end of iteration t will pass to iteration $t+1$;
iv with probability p_a, an egg is removed from the nest and replaced by a new one in a new location.

However, in the cuckoos' parasitic reproduction behavior there is no mechanism that allows the cuckoos to select the best "quality" eggs that survive and therefore it cannot be used to "inspire" rule iii. Indeed, by including this rule as part of the algorithm description, the authors made the cuckoos' parasitic reproduction behavior look as an optimization process, when this is not the case. Also note that the central idea in rule iii is no other than the evolutionary concept of "the survival of the fittest", which was originally introduced to the field of stochastic optimization by the evolutionary computation community and, as we describe in detail below, it is one of several concepts used in cuckoo search that belong to the $(\mu + \lambda)$–evolution strategy (Schwefel 1981).

Differences between algorithm description and algorithm implementation — In (Yang and Deb 2009, 2010), the authors of cuckoo search provided the pseudocode of the algorithm (reported in Algorithm 8) and, in (Yang 2021; Yang and Deb 2010), an example of its correct implementation. In the following, we present the many differences we found between these two components. We do so by comparing Algorithm 8 to steps 1–4 that correspond to the implementation of the algorithm in Matlab—see (Yang 2021) for details.[4]

The first difference to note between steps 1–4 and what is depicted in Algorithm 8, is that, in Algorithm 8, there is not a **for** loop to iterate over all the μ solutions in the population. Therefore, differently from step 2 *(perturbation)*, in

[4]We remind the reader that the code of the algorithm is also publicly available in the *Appendix: Demo Implementation* of (Yang and Deb 2010) in the version published in the arXiv repository: `arXiv:1005.2908`.

Algorithm 8 Cuckoo search algorithm as published in (Yang and Deb 2009, 2010)

1: **begin**
2: Objective function $f(\vec{x})$, $x = (x_1, \ldots, x_d)^T$
3: Initial population of n hosts nests $\vec{x}_i (i = 1, 2, \ldots, n)$
4: **while** ($t <$ MaxGenerations) or (stop creiterion) **do**
5: Get a cuckoo (say i) randomly by Lévy flights
6: Evaluate its quality/fitness F_i
7: Choose a nest among n (say j) randomly
8: **if** ($F_i > F_j$) **then**
9: Replace j by the new solution
10: **end if**
11: Abandon a fraction (p_a) of the worse nests [and build new ones at new locations via Lévy flights]
12: Keep the best solutions (or nests with quality solutions)
13: Rank the solutions and find the current best
14: **end while**
15: Postprocess results and visualization
16: **end**

Algorithm 8, Equation 6.39 is applied only to one solution i randomly selected from the population at every iteration (line 5 of Algorithm 8).

The second difference has to do with step 3 *(selection)*. In this step, after a solution $\vec{x}_t^i$ has been perturbed using Equation 6.39, either the perturbed solution $\vec{x}_t^{i'}$ or the initial solution $\vec{x}_t^i$ is accepted as $\vec{x}_{t'}^i$ depending on its quality—see Equation 6.40. However, in Algorithm 8 this is done differently. In Algorithm 8, the condition in the **if** statement (line 8 of Algorithm 8) says that **if** $f(\vec{x}_t^{i'})$ is better than $f(\vec{x}_t^j)$, where $\vec{x}_t^j$ is a randomly chosen solution, **then** $\vec{x}_t^j$ is replaced by the perturbed solution $\vec{x}_t^{i'}$. Clearly, since j is chosen randomly, it may or may not correspond to i.

The last and most important difference we found concerns step 4 *(recombination)* and the original corresponding algorithm instruction indicated in line 11 of Algorithm 8. First, according to the rules derived from the metaphor by the authors, solutions are supposed to be removed randomly, but in line 11 of Algorithm 8 this is done deterministically. Second, although the authors do not give precise directions on how to implement line 11 of Algorithm 8, from what it is written in this line, it is understood that the solutions are first ranked (otherwise it is not possible to know which ones are the worst) and then Equation 6.39—that is, the equation of the "Lévy flights"—is applied to

the worst $(p_a \times n)$ solutions. However, in the Matlab implementation, line 11 of Algorithm 8 is implemented using Equation 6.41, that recombines two randomly selected solutions and uses them to perturb the solution vector probabilistically and dimension-wise.

It is therefore unclear whether the authors definition of cuckoo search is the one presented in the paper (i.e., Algorithm 8) or in the Matlab implementation. Indeed, in (Yang and Deb 2010, p. 3)[5], they write:

> *"A demo version is attached in the Appendix (this demo is not published in the actual paper, but as a supplement to help readers to implement the cuckoo search correctly)."*

The Cuckoo Search is an ES

We can easily see that cuckoo search is an *evolution strategies*, namely the $(\mu + \lambda)$–ES (Schwefel 1981) with two minor modifications. To explain in detail these two modifications, in Algorithm 9, we report the cuckoo search algorithm as implemented by its authors (see steps 1–4 in Section 6.3.7).

Algorithm 9 Cuckoo search as implemented in (Yang 2021)

```
1:  begin
2:      t ← 0
3:      initialize μ cuckoos (solutions)                          ▷ Eq. 6.38
4:      evaluate μ cuckoos
5:      while not termination-condition do
6:          t ← t + 1
7:          apply mutation to μ cuckoos to create μ eggs (new     ▷ Eq. 6.39
              solutions)
8:          evaluate eggs
9:          select μ solutions from the set of (μ-cuckoos + μ-eggs) ▷ Eq. 6.40
10:         apply recombination to the μ selected solutions       ▷ Eq. 6.41
11:         evaluate the μ selected solutions and use them as cuckoos
              for the next iteration
12:     end while
13: end
```

As we discussed above in Section 6.2, in order to instantiate the $(\mu + \lambda)$–ES, it is necessary to choose the specific type of *parental selection*, *mutation*, *recombination* and *survival selection* to be used. The same is true for cuckoo

[5]This quote is taken from the version of (Yang and Deb 2010) that is published in the arXiv repository in arXiv:1005.2908.

search. In the following, we discuss one by one the way the EC operators have been selected in cuckoo search.

- Parental selection — As it can be seen in lines 3, 7 and 9 of Algorithm 9 and lines 3, 7 and 10 of Algorithm 7, both the $(\mu + \lambda)$–ES and cuckoo search use the same parental selection mechanism. That is, at each iteration, μ parents generate λ offspring, with $\mu = \lambda$.

- Survival selection — In the $(\mu + \lambda)$–ES, survival selection operates over parent-offspring couplings, which means that parents will pass from one generation to another until they are replaced by an offspring with better fitness. As it can be seen in Algorithm 9, line 9 and Equation 6.40, cuckoo search uses the same survival mechanism than the $(\mu + \lambda)$–ES to select between cuckoo–egg couplings.

- Mutation — Both ESs and cuckoo search generate mutations by sampling a random distribution. While cuckoo search uses the Lévy distribution, in ESs, there are a number of possible options: the Gaussian distribution, that was used in the original algorithm (Schwefel 1981); the Cauchy distribution, that was introduced in (Kappler 1996) in 1994; and the Lévy distribution introduced in (Iwamatsu 2002; Lee and Yao 2004) in 2002. Therefore, the mutation operator used by cuckoo search is exactly the same as in ESs.

- Recombination In cuckoo search, recombination—see Algorithm 9, line 10 and Equation 6.41—differs in two aspects from the way this operator is traditionally implemented in ESs.

  The first difference is that the specific recombination mechanism implemented in cuckoo search was originally defined for differential evolution (see Section 6.2). In particular, if we set $\beta = \mathcal{U}[0, 1]$ in Equation 6.11 of DE, the recombination operator proposed in Equation 6.41 becomes the same as the creation of the trial vector in Equation 6.12.

  The second difference is that in cuckoo search recombination is applied at the end of the **while** loop—line 10 of Algorithm 9; differently, in $(\mu + \lambda)$–ES, recombination is applied at the beginning of the loop—line 7 of Algorithm 7. Similarly to ESs, the recombination operator is optional in cuckoo search, since it can be removed from the algorithm implementation by setting parameter $p_a = 0$ in Equation 6.41; in this case cuckoo search

is exactly the same as the $(\mu + \lambda)$–ES. However, when parameter $p_a > 0$, the two algorithms become equivalent starting from iteration 2; in fact, in Algorithm 9 *parental selection* and *recombination* (line 10) are followed by *mutation* (line 7) and then by *survival selection* (line 9). As cuckoo search does not follow the normal order in which the four main components of evolutionary algorithms are used, solutions have to be evaluated twice at each iteration of the **while** loop, which results in wasting computational time.

6.4 Summary

In this chapter, we presented a rigorous, component-based analysis of seven widespread metaphor-based metaheuristics—*grey wolf optimizer* (GWO), *moth-flame algorithm* (MFA), *whale optimization algorithm* (WOA), *firefly algorithm* (FA), *bat algorithm* (BA), *antlion optimizer* (ALO), and *cuckoo search* (CS)—in which we first identified the ideas proposed in each of them, and then, we compared them with those that have been proposed in the context of particle swarm optimization (PSO) and evolution strategies (ESs). We showed that, although the seven metaheuristics were presented as "novel" approaches by their authors, they lack any novelty, as GWO, MFA, WOA, FA, BA are variants of PSO, and ALO and CS are variants of ESs. Also, we evaluated the metaphors that inspired the seven metaheuristics according to the criteria of usefulness and novelty that we defined, and found that none of the metaphors have been useful to develop a novel optimization algorithm, nor is there a sound motivation that justifies their use.

As discussed in chapter 4, the publication of metaphor-based algorithms has detrimental effects to the field of metaheuristics, such as creating confusion in the literature, hindering our understanding of the existing metaheuristic algorithms, and making it troublesome to compare algorithms both conceptually and experimentally. Unfortunately, despite the ample evidence suggesting that the only purpose of framing algorithms into new metaphors is to conceal their similarities with other techniques published before and to make it difficult to see that there is nothing really novel in them, proving that this is the case for all (or a great majority) of them would be very challenging due to the myriad of algorithms of this kind already published and the fact that more appear all too often. Therefore, it is urgent that metaphor-based algorithms are not published

anymore unless their authors (i) present the algorithms using the standard optimization terminology, and (ii) are able to show that the new behavior leads to new ideas that are useful in optimization.

Part III

Designing Metaheuristics Automatically

Chapter 7

Metaheuristic Software Frameworks

A metaheuristic software framework (MSF) is a parameterized software tool that contains the design space of a metaheuristic. In other words, it is a software implementation of the components of a metaheuristic and of the many different ways in which they can be combined to create metaheuristic implementations. To automatically generate a metaheuristic implementation, the MSF is used in combination with an automatic configuration tool (ACT), which, as explained in Chapter 3, iteratively executes the MSF with different configurations and evaluates its performance on a set of problem instances until a configuration for the MSF (which, in this case, represents a concrete metaheuristic implementation) is found that satisfies the needs of the user.

It is important to differentiate between the concepts of automatic configuration and automatic design. The former refers to the task of fine-tuning the parameter values of an already defined metaheuristic design, while the latter refers to composing new metaheuristic designs by recombining their components in new ways in addition to fine-tuning their parameters values. This distinction between automatic configuration and automatic design is useful as they have different objectives. The goal in automatic configuration is to find a well-performing parameter setting for a considered metaheuristic without changing the components of its implementation. On the other hand, in automatic design, the goal is typically to explore new combinations of components and parameter settings never considered before.

7.1 Automatic Design Using MSFs

Most MSFs proposed in the early days, with very few exceptions (such as ParadisEO (Cahon et al. 2004; Keijzer et al. 2002) and (Durillo et al. 2010) that we discuss below), only enabled the use of ACTs to perform automatic configuration tasks. If the users were interested in performing also automatic design tasks using these MSFs, they had to make major adaptations to the code of the MSF in order to extend its metaheuristic design space with new components and rules to combine them. In the worst case scenario, a complete re-implementation of the MSF was necessary. Today, the way the design of MSFs is approached has changed dramatically. Differently from their earlier counterparts, most modern MSFs strive to have a flexible, modular design that allows users to apply them to solve different kinds of problems and to easily extend them with new metaheuristic components and rules to combine them.

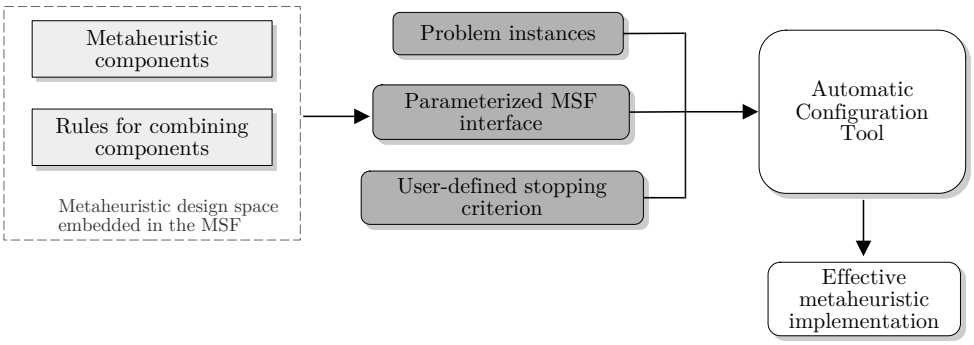


Figure 7.1: General approach for combining modular MSFs and ACTs.

The general approach for combining MSFs with flexible, modular design and the use of ACTs to instantiate ad hoc metaheuristic implementations for specific problems or problem instance distributions is shown in Figure 7.1. The goal of this approach is to enable the automated solution of new problems instances by letting the configuration tool find an effective metaheuristic implementation. Therefore, human intervention is necessary only in those cases in which one wishes to add new metaheuristic components to the MSF; fortunately, this task is typically rather easy thanks to the modular design of the MSFs.

Often, the main challenge in the creation of MSFs is the definition of the rules that control the way the different metaheuristic components can be com-

bined. There are two main ways to do so: *algorithm templates* and *grammar-based programming*. Algorithm templates (or top-down design) consists in creating a parameterized algorithm template in which the metaheuristic components are represented as possible alternatives in a typically fixed algorithmic procedure. Differently, in grammar-based programming (or bottom-up design) the correct combination of components is checked against a grammar, that is, a set of "production rules" that are applied repeatedly. The main difference between the two approaches is that algorithm templates can be much easier to define than grammars but provide limited flexibility in the metaheuristic implementation's design (such as components recursion), whereas using grammar-based programming can be conceptually more difficult but allows to create designs that are much more complex (Mascia et al. 2014).

7.2 Main MSFs Available in the Literature

As shown in Table 1, there are several MSFs proposed in the literature that enable the use of automatic design of metaheuristics. In the table, the MSFs can be categorized by the main kind of metaheuristic components that are included in the software framework and the type of problems they can be used to tackle. Note that the last four MSFs in the table are of a more general nature as they include components from multiple different metaheuristics. It is worth noting that the list in Table 1 is not exhaustive. Indeed, in a recently published paper (Dréo et al. 2021), some of the current maintainers of ParadisEO identified other 47 MSFs that are available on the web. However, many of them are either closed source, unmaintained, and/or were not aimed at designing new metaheuristic implementations (i.e., they only enable the use of automatic configuration and therefore they only optimize the value of numerical parameters). In the following, we discuss some of the MSFs listed in Table 1, paying particular attention to ParadisEO, HeuristicLab, jMetal and EMILI, which currently are some of the most comprehensive and actively maintained.

ACO-TSP-QAP — ACO-TSP-QAP (López-Ibáñez et al. 2017) is MSF for ACO that allows to solve instances of the traveling salesman problem (TSP) and the quadratic assignment problem (QAP). ACO-TSP-QAP provides a unified implementation of the following ACO variants, Ant system (AS), $\mathcal{MAX}$–$\mathcal{MIN}$ ant system ($\mathcal{MM}$AS), Elitist ant system (EAS) (Dorigo 1992b), Rank-based ant

system (RAS) (Bullnheimer et al. 1999a), Ant colony system (ACS) and Best-worst AS (BWAS) (Cordón et al. 2000), while keeping a clear separation between the problem-specific components—such as the representation of the solutions and constraints of the problem—and the ones with general applicability—such as pheromone update rules, restart mechanisms, and local search procedure.

MOACO — The multiobjective ACO software framework (MOACO) (López-Ibáñez and Stützle 2012) allows to tackle multiobjective combinatorial optimization problems. In MOACO, the authors synthesize the metaheuristic components of several multiobjective ACO implementations proposed in the literature into a single generalized algorithm template that allows to combine them in many different ways. Similarly to ACO-TSP-QAP, MOACO keeps a separation between metaheuristic components operating at different levels of the metaheuristic design, particularly between those at a multiobjective level, such as the use of multiple pheromone matrices, and those at a single-objective level, such as the pheromone initialization, updating and evaporation. The goal of separating components in this way is to be able to focus the design process on finding performing combinations of multiobjective components, while delegating the single-objective details to an effective, already designed ACO implementations, such as $\mathcal{MM}$AS or ACS.

UACOR — UACOR (Liao et al. 2014a) is unified software framework for ACO on continuous domains (UACOR) that includes metaheuristic components of three popular ACO variants proposed for the approximate solution of continuous optimization problems, namely ACO$_{\mathbb{R}}$ (Socha and Dorigo 2008), DACO$_{\mathbb{R}}$ (Leguizamón and Coello 2010), and IACO$_{\mathbb{R}}$-LS (Liao et al. 2011). UACOR was build using a modular design that allows the use of automatic configuration tools, in this case `irace`, to generate high-performing continuous ACO implementations. The experimental study of UACOR showed that the automatically generated ACO implementations were, in fact, either competitive or superior than state-of-the-art techniques, such as the popular CMA-ES with incremental population (Auger and Hansen 2005).

ABC-X — ABC-X (Aydın et al. 2017a) is a MSF for the artificial bee colony algorithm (ABC) (Karaboga et al. 2005; Karaboga and Basturk 2007), that was developed by integrating and extending the ideas proposed in a number of ABC variants. The authors of ABC-X defined a generalized mathematical model

Table 7.1: List of some of the most representative MSFs available in the literature that can be used for the automatic design of metaheuristic implementations.

Metaheuristic	Name of the MSF	Type of problem	Num. of objectives	Year	Reference
Ant colony optimization	ACO-TSP-QAP	discrete	single objective	2017	(López-Ibáñez et al. 2017)
Ant colony optimization	MOACO	discrete	multiobjective	2012	(López-Ibáñez and Stützle 2012)
Ant colony optimization	UACOR	continuous	single objective	2014	(Liao et al. 2014a)
Artificial bee colony	ABC-*X*	continuous	single objective	2017	(Aydın et al. 2017b)
Evolutionary computation	ModCMA–ES	continuous	single objective	2021	(Nobel et al. 2021)
Evolutionary computation	DEAP	discrete/continuous	single/multiobjective	2012	(Fortin et al. 2012)
Evolutionary computation	AutoMOEA	discrete/continuous	multiobjective	2015	(Bezerra et al. 2016)
Hybrids of particle swarm optimization and differential evolution	PSO–DE	continuous	single objective	2020	(Boks et al. 2020)
Particle swarm optimization	PSO-*X*	continuous	single objective	2021	(Camacho-Villalón et al. 2022b)
Particle swarm optimization	MOPSO	continuous	multiobjective	2022	(Doblas et al. 2022)
Randomized local search	SATenstein	discrete	single objective	2009	(KhudaBukhsh et al. 2009, 2016)
Multiple metaheuristics	ParadisEO	discrete/continuous	single/multiobjective	2002	(Keijzer et al. 2002) (Cahon et al. 2004) (Dréo et al. 2021)
Multiple metaheuristics	HeuristicLab	discrete/continuous	single/multiobjective	2005	(Wagner and Affenzeller 2005)
Multiple metaheuristics	jMetal	discrete/continuous	single/multiobjective	2010	(Durillo et al. 2010)
Multiple metaheuristics	EMILI	discrete/continuous	single objective	2019	(Pagnozzi and Stützle 2019)

for the main search equation of ABC that allows to instantiate a large number of existing variants and to create many new ones never proposed before. The performance of ABC-X was evaluate on a large set of continuous functions and it was found to be better than the state-of-the-art of ABC.

ParadisEO — ParadisEO (Cahon et al. 2004; Dréo et al. 2021; Keijzer et al. 2002) is a well-known MSF whose initial development dates back to the early 2000's. This MSF includes four main modules that allow users to compose metaheuristic designs, namely: *evolving object* for population-based metaheuristics, *moving objects* for local search algorithms, *estimation of distribution objects* for estimation of distribution algorithms, and *multiobjective evolving objects* for multiobjective optimization. Some of the key features of ParadisEO are: (i) it has a high runtime speed (as it is implemented in `C++`), (ii) it integrates a state-of-the-art benchmarking and profiling tool called IOHprofiler (Doerr et al. 2018) that allows to simplify the process of comparing and evaluating implementations against a benchmark, and (iii) it has an active community of maintainers. ParadisEO has been applied to solve optimization problems for more than two decades. However, in its early days it was manually configured and its use in the context of automatic design is something that has been investigated much more recently in the literature (Dréo et al. 2021). To the best of our knowledge, the first work in this direction is (Aziz-Alaoui et al. 2021), where the authors studied 19 genetic algorithms for the W-model problem that were automatically generated using ParadisEO and `irace`; among the several findings of the authors are that the implementations automatically generated by `irace` were able to outperform all manually created baseline algorithms and that the fast computations that ParadisEO is able to provide allow to tackle large design spaces in short wall-clock times.

HeuristicLab — HeuristicLab (Wagner and Affenzeller 2005) is an optimization software system developed in the early 2000's that incorporates an MSF. In its current version 3.3, released in 2010, the MSF of HeuristicLab has modules for instantiating many different machine learning and metaheuristic algorithms (e.g., genetic programming, evolutionary computation, particle swarm optimization, and simulated annealing). In addition to providing an MSF, HeuristicLab also has a number of useful features: (i) it uses a meta-model that allows to represent arbitrary optimization algorithms; (ii) it allows to manipulate and define metaheuristic designs via a graphical user interface; (iii) it gives easy

access to many problems instances that can be used for benchmarking purposes; and (iv) it provides interactive charts for results analysis. It should be noted that, while ParadisEO and EMILI (described below) are implemented in `C++`, HeuristicLab is implemented in `C#` and is therefore slower. Except for one paper addressing the algorithm selection problem (Beham et al. 2018), we could not find any work specifically targeting the automatic design of metaheuristics using HeuristicLab.

jMetal — jMetal (Durillo et al. 2010), developed in 2009, is an optimization software system implemented in `Java` that incorporates an MSF in addition to several other useful features. jMetal is focused on multiobjective optimization problems, and therefore, it allows to instantiate many state-of-the-art metaheuristics of this kind, such as NSGA-II (Deb et al. 2002), GDE3 (Kukkonen and Lampinen 2005), and IBEA (Zitzler and Künzli 2004). However, in its current version, jMetal also includes components from a number of single-objective algorithms, such as differential evolution, particle swarm optimization, and CMA-ES. The main features of jMetal are that (i) it provides a simple graphical user interface that allows to set the parameters of the metahueristic implementation; (ii) it gives access to five popular testbeds that can be used for benchmarking purposes (e.g., ZDT (Zitzler et al. 2000), DTLZ (Deb et al. 2005), and WFG (Huband et al. 2006)); (iii) it makes available some of the most widely used quality indicators used in multiobjective optimization, namely hypervolume (Zitzler and Thiele 1999), spread (Deb et al. 2002), generational distance (Van Veldhuizen and Lamont 1998), inverted generational distance (Van Veldhuizen and Lamont 1998), and epsilon (Knowles et al. 2006); and (iv) it offers support for performing experimental studies, including the automatic generation of LaTeX tables, statistical pairwise comparison using the Wilcoxon test, and `R` boxplots. There are several examples of the use of jMetal for automatically creating metaheuristic implementations in the literature (see for example (Nebro et al. 2019) and (Doblas et al. 2022)).

EMILI — EMILI (Pagnozzi and Stützle 2019), whose initial development dates back to 2015, is an MSF that implements metaheuristic-specific and problem-specific components for stochastic local search algorithms. In its current version, EMILI is mostly used for single-solution metaheuristics (e.g., iterated local search, tabu search, simulated annealing, etc.), but its design makes it easily extensible to population-based metaheuristics. The main particularity

of EMILI is its architecture, which uses a grammatical representation to validate possible combinations of algorithm components: the components that make up the metaheuristic implementation and the order in which they will be executed are checked against a grammar and then encoded as a character string, so that only valid combinations are produced. Then, EMILI translates the character string into a parametric form that can be executed by an ACT. Other important features of EMILI are: (i) that it implements a strict separation between algorithm-related components and problem-related components, and (ii) that it allows to consider algorithms as recursive metaheuristic components. So far, the two most relevant works using EMILI to automatically create metaheuristic designs and implementations are: (Pagnozzi and Stützle 2019), which is focused on hybrid stochastic local search algorithms for permutation flowshop problems; and (Franzin and Stützle 2019), which is focused on simulated annealing for the quadratic assignment and permutation flowshop problems.

7.3 Summary

The creation of MSFs with flexible, modular design that can be used together with an ACTs is a promising research direction in the field of metaheuristics. The goal of this approach is twofold: first, to reduce the need for manual (human) intervention in the design process of metaheuristic implementation; and second, to allow users to easily instantiate high-performing, ad hoc implementations for specific problems or problem instance distributions. In the next chapter, we describe in detail PSO-*X*, which is a MSF for particle swarm optimization that was created in the context of this research work. PSO-*X* has a design that is similar to UACOR and ABC-X, and is currently the most complete MSF proposed in the literature for PSO.

Chapter 8

PSO-*X*: A Flexible, Modular Framework for Particle Swarm Optimization

Particle swarm optimization (PSO) has been the object of many studies and modifications for more than 25 years. Ranging from small refinements to the incorporation of sophisticated novel ideas, the majority of the modifications proposed to this metaheuristic have been the result of a manual process in which developers try new designs based on their own knowledge and expertise. However, as discussed in great detail in Chapter 3, manually introducing changes is very time consuming and makes the systematic exploration of the metaheuristic design space a difficult process.

In this chapter, we present PSO-*X*, a metaheuristic software framework (MSF) specifically designed to integrate the use of automatic configuration tools (ACTs) into the process of generating PSO implementations. Our framework embodies a large number of metaheuristic components developed over more than 25 years of research that have allowed PSO to deal with a large variety of problems, and uses `irace`, a state-of-the-art configuration tool, to automatize the task of selecting and configuring PSO implementations starting from these components. We show that `irace` is capable of finding high performing instances of PSO implementations never proposed before.

8.1 Preliminaries

8.1.1 Main Concepts of PSO

Particle swarm optimization (Eberhart and Kennedy 1995; Kennedy and Eberhart 1995) is a randomized optimization algorithm where a set of "particles" search for approximate solutions of continuous optimization problems. In PSO each particle moves in the search space by repeatedly applying *velocity* and *position* update rules. Each particle i has, at every iteration t, three associated vectors: the position $\vec{x}_t^i$, the velocity $\vec{v}_t^i$, and the personal best position $\vec{p}_t^i$. The vector $\vec{x}_t^i$ is a candidate solution to the optimization problem considered whose quality is evaluated by the objective function $f(\cdot)$.

In addition to these vectors, each particle i has a set N^i of neighbors and a set I^i of informants. Set N^i contains the particles from which i can obtain information, whereas $I^i \subseteq N^i$ contains the particles that will indeed provide the information used when updating i's velocity. The way the sets N^i—which define the *topology* of the swarm (Kennedy and Mendes 2002)—and the sets I^i—which we refer to as *models of influence*—are defined are two important design choices in PSO. Sets N^i can be defined in many different ways producing a large number of possible different topologies; the two extreme cases are the fully-connected topology, in which all particles are in the neighborhood of all other particles, and the ring topology, where each particle is a neighbor of just two adjacent particles. Examples of other partially connected topologies include lattices, wheels, random edges, etc. The model of influence can also be defined in different ways, but the vast majority of implementations employ either the *best-of-neighborhood* which contains the particle with the best personal best solution in the neighborhood of i (which includes particle i itself), or the *fully informed* model, in which $I^i = N^i$.

In the standard PSO (StaPSO) (Shi and Eberhart 1998), the rule used to update particles' position is

$$\vec{x}_{t+1}^i = \vec{x}_t^i + \vec{v}_{t+1}^i, \tag{8.1}$$

where the velocity vector $\vec{v}_{t+1}^i$ of the i^{th} particle at iteration $t+1$ is computed using an update rule that involves $\vec{v}_t^i$, $\vec{p}_t^i$, and $\vec{l}_t^i$. The vector $\vec{l}_t^i$ indicates the best among the personal best positions of the particles in the neighborhood of i; formally, it is equal to $\vec{p}_t^k$ where $k = \arg\min_{j \in N^i}\{f(\vec{p}_t^j)\}$. Note that when a

fully-connected topology is employed, vector $\vec{l}_t^i$ becomes the *global best* solution and is indicated as $\vec{g}_t$.

The velocity update rule of StaPSO is defined as follows:

$$\vec{v}_{t+1}^i = \omega \vec{v}_t^i + \varphi_1 U_{1t}^i (\vec{p}_t^i - \vec{x}_t^i) + \varphi_2 U_{2t}^i (\vec{l}_t^i - \vec{x}_t^i), \tag{8.2}$$

where ω is a parameter, called inertia weight, used to control the influence of the previous velocity, and φ_1 and φ_2 are two parameters known as the acceleration coefficients that control the influence of $(\vec{p}_t^i - \vec{x}_t^i)$ and $(\vec{l}_t^i - \vec{x}_t^i)$. The goal of vectors $(\vec{p}_t^i - \vec{x}_t^i)$ and $(\vec{l}_t^i - \vec{x}_t^i)$, respectively known as the cognitive influence (CI) and the social influence (SI), is to attract particles towards high quality positions found so far. U_{1t}^i and U_{2t}^i are two $d \times d$ diagonal matrices whose diagonal values are random values drawn from $\mathcal{U}(0, 1]$; their function is to induce perturbation to the CI and SI vectors.

The rule to update the personal best position of particle i is

$$\vec{p}_{t+1}^i = \begin{cases} \vec{x}_{t+1}^i, & \text{if } ((f(\vec{x}_{t+1}^i) < f(\vec{p}_t^i)) \ \wedge \ (\vec{x}_{t+1}^i \in S)) \\ \vec{p}_t^i & \text{otherwise.} \end{cases} \tag{8.3}$$

8.1.2 Previous Work on the Automatic Design of PSO

Compared to the number of works devoted to other widely used metaheuristics, such as ant colony optimization (López-Ibáñez and Stützle 2012), evolutionary computation (Bezerra et al. 2016; Nobel et al. 2021), and artificial bee colony (Aydın et al. 2017b), there are very few previous work attempting the automatic design of particle swarm optimization implementations. The two most relevant of these works are (Miranda and Prudêncio 2015; Poli et al. 2005), where the authors used grammatical evolution (GE) to evolve novel velocity update rules in PSO. The main limitation of these works is the low number of different components that can be combined. In (Poli et al. 2005), only the social and cognitive components of the velocity update rule can be automatically designed; while, in (Miranda and Prudêncio 2015), the list of components includes also the topology and swarm size, but the grammar that defines the rules to combine components is based on the standard version of PSO and makes difficult to include recent metaheuristic components.

To overcome these limitation, we propose PSO-*X*, a flexible, component-based framework containing a large number of metaheuristic components

previously proposed in the PSO literature. In PSO-*X* each metaheuristic component can assume a set of different values and PSO-*X* generates a specific PSO implementations by selecting a value for each possible component. To do so, PSO-*X* uses a generalized algorithm template that is flexible enough to combine the metaheuristic components in many different ways, and that is sufficient to synthesize many well-known PSO variants published in the last two decades. Most of the flexibility in PSO-*X* is achieved through the use of a generalized velocity update rule—the core component of PSO. The goal of using a generalized velocity rule is to facilitate the abstraction of the elements typically used in this metaheuristic component, in order to allow the combination of concepts that operate at different levels of the metaheuristic implementation design. For example, with our template and the generalized velocity update rule, a high-level component such as the type of distribution of all next possible particle positions can interact with specific types of perturbation and a number of strategies to compute their magnitude.

PSO-*X* provides two important benefits when implementing PSO: first, the possibility of easily creating many different implementations combining a wide variety of metaheuristic components from a single framework; second, the possibility of using automatic configuration tools to tailor implementations of PSO to specific problems according to different scenarios. Our aim is to show that developing PSO implementations using PSO-*X* is more efficient and produces implementations capable of outperforming their manually designed counterparts. To assess the effectiveness of our PSO-*X* framework, we compare the performance of six automatically generated PSO implementations with ten of the best known variants proposed in the literature over a set of fifty benchmark problems for evaluating continuous optimizers.

8.1.3 Automatic Algorithm Configuration Using `irace`

We employed a state-of-the-art offline configuration tool called `irace` (López-Ibáñez et al. 2016). This tool has been shown to be capable of dealing with the task of selecting, configuring and generating high-performing metaheuristic implementations by finding good configurations whose performance can be generalized to unseen problem instances. To do so, `irace` implements a procedure called *iterated racing* (López-Ibáñez et al. 2016), which is based on the machine learning model selection approach called racing (Maron and Moore 1997) and on Friedman's non-parametric two-way analysis of variance. Iterated racing

consists of the following steps. First, it samples candidate configurations from the parameter space. Second, it evaluates the candidate configurations on a set of instances by means of races, whereby each candidate configuration is run on one instance at a time. Third, it discards the statistically worse candidate configurations identified using a statistical test based on Friedman's non-parametric two-way analysis of variance by ranks. During the configuration process, which is done sequentially and uses a given computational budget, `irace` adjusts the sampling distribution in order to bias new samplings towards the best configurations found so far. When the computational budget is over, `irace` returns the configuration that performed best over the set of training instances. `irace` is capable of handling the different types of parameters included in our framework, that is, numerical (e.g., ω, φ_1 or φ_2,), categorical (e.g., *topology*), and subordinate parameters, that is, parameters that are only necessary for particular values of other parameters (e.g., when the size of the population changes in the implementation, it is necessary to configure the maximum and minimum number of particles in the swarm, but not when the size remains constant.)

8.2 Design Choices in PSO

Many metaheuristic components have been proposed for PSO over the years (Bonyadi and Michalewicz 2017; Poli et al. 2007) with the goal of improving its performance and enabling its application to a wider variety of problems. We have categorized these metaheuristic components into five different groups: (1) those used to set the value of the main control parameters, (2) those that control the distribution of particles positions in the search space, (3) those used to apply perturbation to the velocity and/or position vectors, (4) those regarding the construction and application of the random matrices, and (5) those related to the topology, model of influence and population size.

Group (1) comprises the time-varying and adaptive/self-adaptive *parameter control strategies* used to compute the value of ω, φ_1 and φ_2. Time-varying strategies take place at specific iterations of the implementation's execution; while adaptive and self-adaptive strategies use information related to the optimization process (e.g., particles average velocity, convergence state of the algorithm, average quality of the solutions found, etc.) to adjust the value of the parameters. Because the value of ω, φ_1 and φ_2 heavily influences the

exploration/exploitation behavior of PSO, parameter control strategies for this metaheuristic are abundant in literature (Harrison et al. 2016). In particular, a lot of attention has been given to control strategies focused on adjusting the value of ω, which is intrinsically related to the local convergence of the algorithm.[1]

Locally convergent implementations not only guarantee to find a local optimum in the search space, but also prevent issues such as (i) *swarm explosion*, which happens when a particle's velocity vector grows too large and the particle becomes incapable of converging to a point in the search space (Clerc and Kennedy 2002); and (ii) *poor problem scalability*, which means that the algorithm performs poorly on high dimensional problems (Bonyadi and Michalewicz 2014). In fact, the poor problem scalability issue has become very relevant in the last years because of the increasing number of problems involving large dimensional spaces where PSO is applicable. It has been observed that unwanted particles roaming in high dimensional spaces is a substantial part of this issue and that, in variants such as StaPSO, parameter values for ω, φ_1 and φ_2 that perform well in low dimensional spaces will most likely perform poorly in large dimensional ones (Oldewage et al. 2020). A number of strategies have been proposed to address this issue, such as reinitialization (García-Nieto and Alba 2011), group-based random diagonal matrices (Zyl and Engelbrecht 2016) and perturbation mechanisms (Bonyadi and Michalewicz 2014).

In group (2) are the metaheuristic components used to control the distribution of all next possible positions (DNPP) of the particles. The chosen DNPP determines the way particles are mapped from their current position to the next one. We consider the three main DNPP proposed in the literature—the rectangular (used in StaPSO), the spherical (used in SPSO-2011 (Clerc 2011)) and the additive stochastic, which comprises the recombination operators proposed for simple dynamic PSO implementations (Peña 2008b). Although some DNPP mappings suffer from *transformation variance*—which happens when the algorithm performs poorly under mathematical transformations of the objective function, such as scale, translation, and rotation[2]—there are a number of metaheuristic components that have been developed to prevent this issue.

Group (3) is composed of the metaheuristic components that allow to apply

[1] In this context, convergence means that, as the number of iterations grows larger, the probability of particles reaching a stable position in which $\vec{x}_t^{\,i} = \vec{p}_t^{\,i} = \vec{g}_t$ and $\vec{V}_t^{\,i} = 0$ for all i approaches 1. See (Bonyadi and Michalewicz 2017; Trelea 2003).

[2] Scale variance means that the performance of the algorithm is affected by uniformly scaling all variables of the problem, while *translation* and *rotation variance* mean that the performance of the algorithm is dependent on how the coordinated axes are placed in the search space.

perturbations to the particles velocity/position vectors. In general, in PSO perturbation mechanisms can be *informed* or *random*. Informed perturbation mechanisms receive a position vector as an input (typically $\vec{p}_t^{\,i}$ or $\vec{x}_t^{\,i}$) and use it to compute a new vector that replaces the one that was received. The typical way in which informed mechanisms work is by using the components of the input vector as center of a probability distribution and mapping random values around them; however, other options found in the literature include computing the Hadamard product between the input vector and a random one, or randomly modifying the components of the input vector. Differently, random perturbation mechanisms add a random value to a particle's position or velocity. Perturbation mechanisms proposed for PSO are used to improve the diversity of the solutions (Bonyadi and Michalewicz 2014; Xinchao 2010), avoid stagnation (Lehre and Witt 2013), and avoid divergence (Bergh and Engelbrecht 2002). Additionally, some of these mechanisms allow to modify the DNPP of the particles; an example is the mechanism proposed in (Bonyadi and Michalewicz 2014), where a Gaussian distribution is used to map random points on spherical surfaces centered around the position of the informants.

One of the main challenges in most perturbation mechanisms is to determine the perturbation magnitude: a strong perturbation may prevent particles from efficiently exploiting high quality areas of the search space, while a weak one may not produce any improvement at all. In order to allow convergent implementations to take advantage of the perturbation mechanism, some magnitude control strategies take into account the state of the optimization process to adjust the magnitude at run time. An example is (Bergh and Engelbrecht 2002), where a parameter decreases the perturbation magnitude when the best solution found so far has been constantly improving, whereas increases it when the algorithm is stagnating. Another example is (Bonyadi and Michalewicz 2014), where the magnitude is computed based on the Euclidean distance between the particles so as to decrease it as particles converge to the best solution found so-far.

The metaheuristic components in group (4) corresponds to the *random matrices*, whose function, similarly to some perturbation mechanism of group (3), is to provide diversity to particles movement. The main difference between the random matrices and the perturbation mechanisms described above is that the former can be used to produce changes in the magnitude and direction of the CI and SI vectors, while the latter allow only to apply perturbation to individual positions used in the computation of the CI and SI. In StaPSO, the random

matrices (U_1^i and U_2^i, see Equation 8.2) are usually constructed as diagonal matrices with values drawn from $\mathcal{U}(0,1)$; however, in some implementations of StaPSO (e.g., (Bonyadi and Michalewicz 2017)), the matrices are replaced by two random values r_1^i and r_2^i—in this case particles oscillate linearly between $\vec{p}_t^{\,i}$ and $\vec{l}_t^{\,i}$ without being able to move in different directions, preventing transformation variance (Bonyadi and Michalewicz 2017) but affecting the performance of the algorithm. Using random rotation matrices[3] (RRMs), instead of random diagonal matrices, is another way to address transformation variance in PSO. RRMs allow to apply random changes to the length and direction of the vectors in the velocity update rule without being biased towards some particular reference frame. The two main methods that have been used to create RRMs in the context of PSO are exponential map (Wilke et al. 2007) and Euclidean rotation (Bonyadi et al. 2014).

The last group of metaheuristic components we identified in our work, group (5), includes the *topology*, *model of influence* and *population size*. The topology plays an important role in the way PSO modulates its exploration-exploitation capabilities. In addition to the well-known fully-connected, ring and von Neumann topologies, there are other topologies that have been explored in the PSO literature, such as hierarchical and small-world network. In (Montes de Oca et al. 2009), a topology that decreases connectivity over time was proposed. Concerning the model of influence, besides the best-of-neighborhood and the fully informed, another option is the ranked fully informed model of influence (Jordan et al. 2008), in which the contribution of each informant is weighted according to its rank in the neighborhood. Concerning population size, it has recently been proposed to increase or decrease the number of particles according to some metrics (Hsieh et al. 2008; Montes de Oca et al. 2010). The number of particles in the swarm has an impact on the trade-off between solution quality and speed of the implementation (Bonyadi and Michalewicz 2017; Montes de Oca et al. 2010). In general, a large population should be used as it can produce better results. However, a small population may be the best option when the objective function evaluation is expensive or when the number of possible function evaluations is limited.

[3] A rotation matrix is an orthogonal matrix whose determinant is 1.

8.3 Designing PSO Implementations From an Algorithm Template

In this section, we explain the way in which the metaheuristic components reviewed in the previous section can be combined using the PSO-*X* framework. In the reminder of the chapter, we use Sans Serif font to indicate the name of the metaheuristic components and of their options as implemented in PSO-*X*.

8.3.1 Algorithm Template for Designing PSO Implementations

Algorithm 10 Algorithm template used by PSO-*X*

Require: set of parameters
1: swarm ← INITIALIZE(Population, Topology, Model of influence)
2: **repeat**
3: **for** $i \leftarrow 1$ **to** size(swarm) **do**
4: $\vec{v}^i_{t+1} \leftarrow \omega_1 \vec{v}^i_t + \omega_2 \,\mathsf{DNPP}(i,t) + \omega_3 \,\mathsf{Pert}_{\mathsf{rand}}(i,t)$
5: apply velocity clamping ▷ optional
6: $\vec{x}^i_{t+1} \leftarrow \vec{x}^i_t + \vec{v}^i_{t+1}$
7: **end for**
8: **for** $i \leftarrow 1$ **to** size(swarm) **do**
9: compute $f(\vec{x}^i_t)$
10: update $\vec{p}^i_t$ using Equation 8.3
11: **end for**
12: apply stagnation detection, particles reinitialization ▷ optional
13: **if** type(Population) $\neq$ Pop-constant **then**
14: swarm ← UPDATEPOPULATION(swarm, Population)
15: **end if**
16: **if** type(Topology) = Top-time-varying **or** type(Population) $\neq$ Pop-constant **then**
17: swarm ← UPDATETOPOLOGY(swarm, Topology, Model of influence)
18: **end if**
19: **until** termination criterion is met
20: **return** *global best solution*

Algorithm 10 depicts the PSO-*X*'s algorithm template. A swarm of particles (swarm) is created using the INITIALIZE() procedure that assigns to each particle a set N^i, a set I^i, an initial position and an initial velocity based on the Population, Topology and Model of influence indicated by the framework user. Additionally, the INITIALIZE() procedure creates and initializes any variable required to use the metaheuristic components included in the implementation. The two **for** cycles

of lines 3 to 7 and lines 8 to 11 corresponds to the standard implementation of PSO—except for line 4 that shows our generalized velocity update rule (GVUR), defined as follows:

$$\vec{v}^{\,i}_{t+1} = \omega_1 \, \vec{v}^{\,i}_{t} + \omega_2 \, \mathsf{DNPP}(i,t) + \omega_3 \, \mathsf{Pert}_{\mathsf{rand}}(i,t), \tag{8.4}$$

where DNPP represents the type of mapping from a particle's current position to the next one, and $\mathsf{Pert}_{\mathsf{rand}}$ represents an additive perturbation mechanism.

The parameter ω_1 is the same as the inertia weight in StaPSO (see Equation 8.2) and its value can be computed using the strategies that have been developed for this purpose (the list of the strategies available to compute its value are shown in Table B.4 of Appendix B). The parameters ω_2 and ω_3 control the influence that will be given to the DNPP and $\mathsf{Pert}_{\mathsf{rand}}$ components; their values can be set equal to ω_1 or be computed using the random component, where $\omega_2, \omega_3 \sim \mathcal{U}[0.5, 1]$, or the constant component, where ω_2, ω_3 are user selected constants in the interval $[0, 1]$. We use three independent ω parameters so that it is easy to disable any of the GVUR components. For example, a velocity free PSO can be easily obtained by setting $\omega_1 = 0$. After all particles have updated their position, two procedures can take place: UpdatePopulation(), that increases/decreases the size of the swarm according to the type of Population employed; and UpdateTopology(), that connects newly added particles to a set of neighbors, or disconnect particles when the topology connectivity reduces over time.

All implementations that can be created using Algorithm 10 and combining metaheuristic components with different functionalities, such as different DNPPs or topologies, are considered valid implementations by PSO-*X*. This allows enough flexibility to explore many new designs without increasing too much the computational complexity of the implementations. In particular, in PSO-*X*, we do not allow the recursive use of components, as it is done in component-based framework using grammars (Stützle and López-Ibáñez 2019). In Table B.2 of Appendix B we show the metaheuristic components and parameters in the framework, their domain and type, and the condition(s) under which each parameter is used in PSO-*X*.

8.3.2 DNPP Component

The six options defined in PSO-X for the DNPP component are DNPP-rectangular, DNPP-spherical, DNPP-standard, DNPP-discrete, DNPP-Gaussian and DNPP-Cauchy–Gaussian.

The DNPP-rectangular option is defined as follows:

$$\text{DNPP-rectangular} = \sum_{k \in I_t^i}^{n} \varphi_t^k \text{Mtx}_t^k \big(\text{Pert}_{\text{info}}(\vec{p}_t^{\,k}) - \vec{x}_t^{\,i}\big), \tag{8.5}$$

where Mtx and $\text{Pert}_{\text{info}}$ are, as mentioned before, high-level representations of the different types of random matrices and informed perturbation mechanisms used in PSO. The DNPP-rectangular is by far the most commonly used in implementations of PSO, including StaPSO (Shi and Eberhart 1998), the constriction coefficient PSO (Clerc and Kennedy 2002), the fully informed PSO (Mendes et al. 2004), etc. In the standard application of DNPP-rectangular (i.e., as in StaPSO), each term added in Equation 8.5 is a vector located on a hyper-rectangular surface whose side length depends on the distance between $\vec{p}_t^{\,k}$ and $\vec{x}_t^{\,i}$. However, when the perturbation component of DNPP-rectangular is an informed Gaussian—as in the locally convergent rotationally invariant PSO (LcRPSO) (Bonyadi and Michalewicz 2014)—or a random rotation matrix (RRM)—as in the diverse rotationally invariant PSO (DvRPSO) (Wilke et al. 2007)—the surface on which the different vectors computed in Equation 8.5 are located becomes hyperspherical or semi-hyperspherical, respectively.

Another option is DNPP-spherical (Clerc 2011), where a vector located on a hyper-sphere is used in the computation of a particle new position. The equation to compute the DNPP-spherical option is the following:

$$\text{DNPP-spherical} = \mathcal{H}_i\big(\vec{c}_t^{\,i}, |\vec{c}_t^{\,i} - \vec{x}_t^{\,i}|\big) - \vec{x}_t^{\,i}, \tag{8.6}$$

where $\mathcal{H}(\vec{c}_t^{\,i}, |\vec{c}_t^{\,i} - \vec{x}_t^{\,i}|)$ is a random point drawn from a hyperspherical distribution with center $\vec{c}_t^{\,i}$ and radius $|\vec{c}_t^{\,i} - \vec{x}_t^{\,i}|$. The center $\vec{c}_t^{\,i}$ is computed as follows:

$$\vec{c}_t^{\,i} = \frac{\vec{x}_t^{\,i} + \vec{L}_t^{\,i} + \vec{P}_t^{\,i}}{3}, \tag{8.7}$$

where

$$\vec{P}_t^{\,i} = \vec{x}_t^{\,i} + \varphi_{1t} \text{Mtx}_t \big(\text{Pert}_{\text{info}}(\vec{p}_t^{\,i}) - \vec{x}_t^{\,i}\big), \tag{8.8}$$

$$\vec{L}_t^i = \vec{x}_t^i + \sum_{k \in I_t^i \setminus \{i\}}^{n} \varphi_{2t}^k \mathsf{Mtx}_t^k \left(\mathsf{Pert}_{\mathsf{info}}(\vec{p}_t^k) - \vec{x}_t^i \right), \tag{8.9}$$

and $\varphi_{2t}^k = \frac{\varphi_{2t}}{|I_t^i \setminus \{i\}|}$. The main difference between DNPP-spherical and the standard implementation of DNPP-rectangular is that the hypersphere $\mathcal{H}_i(\vec{c}_t^i, |\vec{c}_t^i - \vec{x}_t^i|)$ is invariant to rotation around its center, whereas DNPP-rectangular is rotation variant unless another component is used to overcome this issue—e.g., a Gaussian perturbation, as done in the LcRPSO variant. While the DNPP-spherical and the LcRPSO combining the DNPP-rectangular with a Gaussian perturbation component use the same idea, they work in a different way. In the DNPP-spherical DNPP, there is a single vector mapped randomly in the hypersphere $\mathcal{H}(\vec{c}_t^i, |\vec{c}_t^i - \vec{x}_t^i|)$ and the informants of i participate only in the computation of vector $\vec{L}_t^i$ (see Equation 8.9)[4]; whereas in the LcRPSO variant, there are n different vectors, one for each informant of i, each mapped on a spherical surface, and the new velocity of the particle is obtained by adding all n vector, as shown in Equation 8.5.

The DNPP-standard, DNPP-discrete, DNPP-Gaussian and DNPP-Cauchy–Gaussian options belong to the class of simple dynamic PSO implementations (Li and Yao 2011; Peña 2008b) and have the form $\vec{q}_t^i - \vec{x}_t^i$, where vector $\vec{q}_t^i$ is computed differently in each option:

$$\text{DNPP-standard}: \ \vec{q}_t^i = \frac{\varphi_1 \vec{p}\,'^i_t + \varphi_2 \vec{p}\,'^k_t}{\varphi_1 + \varphi_2} \tag{8.10}$$

$$\text{DNPP-discrete}: \ \vec{q}_t^i = \eta_d \, \vec{p}\,'^i_t + (1 - \eta_d) \, \vec{p}\,'^k_t) \tag{8.11}$$

$$\text{DNPP-Gaussian}: \ \vec{q}_t^i = \mathcal{N}(\frac{\vec{p}\,'^i_t + \vec{p}\,'^k_t}{2}, |\, \vec{p}\,'^i_t - \vec{p}\,'^k_t|) \tag{8.12}$$

$$\text{DNPP-Cauchy–Gaussian}: \ \vec{q}_t^i = \begin{cases} p_t^{i,j} + \mathcal{C}(1)|p_t'^{i,j} - p_t'^{k,j}| & \text{if } \mathcal{U}[0,1] \leq r \\ p_t^{k,j} + \mathcal{N}(0,1)|p_t'^{i,j} - p_t'^{k,j}| & \text{otherwise} \end{cases} \tag{8.13}$$

where $\eta_d \sim \mathcal{U}\{0,1\}$ is a discrete random number drawn from a Bernoulli distribution, $\mathcal{C}(1)$ is a random number generated using a Cauchy distribution with scaling parameter 1, $\mathcal{N}(0,1)$ is a random number from a Normal distribution with mean 0 and variance 1, and r is a parameter that allows the user to select the probability with which the Cauchy or the Normal distributions are used in

[4] In the original definition of Equation 8.9, vector $\vec{L}_t^i$ was defined considering a best-of-neighborhood model of influence. In this article, we have extended the computation of $\vec{L}_t^i$ to an arbitrary number of informants.

Equation 8.13. Vectors $\vec{p}_t'^{\,i}$ and $\vec{p}_t'^{\,k}$ are computed using $\vec{p}_t'^{\,i} = \mathsf{Pert}_{\mathsf{info}}(\vec{p}_t^{\,i})$ and $\vec{p}_t'^{\,k} = \mathsf{Pert}_{\mathsf{info}}(\vec{p}_t^{\,k})$ with $k \in I^i$.

Unlike options DNPP-standard, DNPP-discrete and DNPP-Gaussian, where the mapping between particles i and k is deterministic, in DNPP-Cauchy–Gaussian, the value of the j^{th} dimension of $\vec{q}_t^{\,i}$ is computed with probability r using $\vec{p}_t^{\,i}$ and a Cauchy distribution; and with probability $1-r$ using $\vec{p}_t^{\,k}$ and a Normal distribution. Although we kept the original definition of these DNPPs for the most part, we did two modifications: we included the $\mathsf{Pert}_{\mathsf{info}}$ component (i.e., vectors $\vec{p}_t'^{\,i}$ and $\vec{p}_t'^{\,k}$ instead of $\vec{p}_t^{\,i}$, $\vec{p}_t^{\,k}$) and the possibility of using a random informant model of influence (Mol-random informant), which consists in choosing a random particle from N^i and use it as informant.

8.3.3 $\mathsf{Pert}_{\mathsf{rand}}$ and $\mathsf{Pert}_{\mathsf{info}}$ Components

The two types of perturbation components included in PSO-X are: $\mathsf{Pert}_{\mathsf{info}}$, which modifies an input vector; and $\mathsf{Pert}_{\mathsf{rand}}$, which generates a random vector that is added to the velocity vector. $\mathsf{Pert}_{\mathsf{info}}$, as explained in Section 8.3.2, is a component used by the DNPP component. Differently, $\mathsf{Pert}_{\mathsf{rand}}$ is used directly in the generalized velocity update rule.

Table 8.1: Options for computing $\mathsf{Pert}_{\mathsf{info}}$ and $\mathsf{Pert}_{\mathsf{rand}}$ components in PSO-X when they are used in the implementation

Component	Option	Definition
$\mathsf{Pert}_{\mathsf{info}}$ *	none	—
	$\mathsf{Pert}_{\mathsf{info}}$-Gaussian	$\mathcal{N}(\vec{r}, \sigma_t)$
	$\mathsf{Pert}_{\mathsf{info}}$-Lévy	$L_{\gamma_t}(\vec{r}, \sigma_t)$
	$\mathsf{Pert}_{\mathsf{info}}$-uniform	$\vec{r} + (\vec{s} \odot \vec{r})$, with $\vec{s} \sim \mathcal{U}[-b_t, b_t]$
$\mathsf{Pert}_{\mathsf{rand}}$ **	none	—
	$\mathsf{Pert}_{\mathsf{rand}}$-rectangular	$\tau_t\,(1 - 2 \cdot \mathcal{U}(0,1))$
	$\mathsf{Pert}_{\mathsf{rand}}$-noisy	$\mathcal{U}[-\delta_t/2, \delta_t/2]$

* In the options for computing $\mathsf{Pert}_{\mathsf{info}}$: $\vec{r}$ is the input vector; $\mathcal{N}(\vec{r}, \sigma_t)$ is a Normal distribution with mean $\vec{r}$ and variance σ_t; $L_\gamma(\vec{r}, \sigma_t)$ is a Lévy distribution with mean $\vec{r}$, variance σ_t, and scale parameter γ_t; and b_t is a real parameter.

** In the options for computing $\mathsf{Pert}_{\mathsf{rand}}$: τ_t and δ_t are two real parameters.

As shown in Table 8.1, both $\mathsf{Pert}_{\mathsf{info}}$ and $\mathsf{Pert}_{\mathsf{rand}}$ are optional components in PSO-X that can be omitted from the implementation using the none option. The options for $\mathsf{Pert}_{\mathsf{info}}$, when the component is present in the implementation, are

$\text{Pert}_{\text{info}}$-Gaussian, $\text{Pert}_{\text{info}}$-Lévy, and $\text{Pert}_{\text{info}}$-uniform. $\text{Pert}_{\text{info}}$-Gaussian and $\text{Pert}_{\text{info}}$-Lévy compute a random vector by using a probability distribution whose center and dispersion are given by the input vector $\vec{r}$ and by the parameter σ_t that controls the magnitude of the perturbation. Similarly, in $\text{Pert}_{\text{info}}$-uniform, the perturbation magnitude depends on a parameter b_t, that controls the interval in which a random vector $\vec{s}$ will be generated using a uniform distribution. Regarding the $\text{Pert}_{\text{rand}}$ component, both $\text{Pert}_{\text{rand}}$-rectangular and $\text{Pert}_{\text{rand}}$-noisy employ a random uniform distribution to generate a random vector; the magnitude of the perturbation is controlled in this case by parameters τ_t and δ_t, respectively.

In $\text{Pert}_{\text{info}}$-Lévy, the value of γ_t can be used to switch between a Gaussian and a Cauchy distribution (Richer and Blackwell 2006). That is, when $\gamma_t = 1$, the Lévy distribution is equivalent to the Gaussian distribution, and when $\gamma_t = 2$, it is equivalent to the Cauchy distribution. In PSO-X, the value of γ_t is obtained sampling from the discrete uniform distribution $\mathcal{U}\{10, 20\}$:

$$\gamma_t = \mathcal{U}\{10, 20\}/10.$$

This allows to vary the probability of generating a random value in the tail of the distribution. This way of computing the value of γ_t is similar to the one used in (Li and Yao 2011) for computing the DNPP-Cauchy–Gaussian option assuming $r = 0.5$ to give the same probability to each case—see Equation 8.13.

Since the perturbation magnitude (PM) plays a critical role in the effectiveness of perturbation components, setting its value (either offline or at run-time) is often challenging. In PSO-X we implemented four strategies for computing the PM that can be used with any of the $\text{Pert}_{\text{info}}$ and $\text{Pert}_{\text{rand}}$ components. These strategies are PM-constant value, PM-Euclidean distance, PM-obj.func. distance, and PM-success rate.

The PM-constant value strategy (Xinchao 2010) is the simplest and consists in using a value that remains constant during the execution of the implementation. This strategy guarantees that the perturbation magnitude is always greater than zero—a condition that has to be verified for all perturbation strategies. However, the main problem with the PM-constant value strategy is that using the same value may not be effective for the different stages of the optimization process. For example, particles that are farther away from the global best solution may benefit from a large PM value in order to move to higher quality areas, while for those particles that are near the global best solution, a small PM value would

make exploitation easier.

The PM-Euclidean distance strategy (Bonyadi and Michalewicz 2014) consists in using the Euclidean distance between the current position of particle i and the personal best of a neighbor k. This strategy is defined as follows:

$$\text{PM}_t^{i,k} = \begin{cases} \epsilon \cdot \text{PM}_{t-1}^{i,k} & \text{if } \vec{x}_t^i = \vec{p}_t^k \\ \epsilon \cdot \sqrt{\sum_{j=1}^{d} (\vec{x}_t^{i,j} - \vec{p}_t^{k,j})^2} & \text{otherwise} \end{cases}, \tag{8.14}$$

where $0 < \epsilon \leq 1$ is a parameter used to weigh the distance between $\vec{x}_t^i$ and $\vec{p}_t^k$.

The PM-obj.func. distance is very similar to the PM-Euclidean distance, but the distance between particles is measured in terms of the quality of the solutions. The equation to compute the PM using PM-obj.func. distance is

$$\text{PM}_t^i = \begin{cases} m \cdot \text{PM}_{t-1}^i & \text{if } \vec{p}_t^i = \vec{l}_t^i \\ m \cdot \frac{f(\vec{l}_t^i) - f(\vec{x}_t^i)}{f(\vec{l}_t^i)} & \text{otherwise} \end{cases}, \tag{8.15}$$

where $0 < m \leq 1$ is a parameter. For particles whose quality is very similar to that of the local best, the PM will be small, enhancing exploitation; and for those whose quality is poor compared to that of the local best, the PM will be large allowing them move to far areas of the search space.

The mechanism implemented in PM-success rate (Bergh and Engelbrecht 2002) to compute the PM takes into account the success rate of the swarm in terms of improving the best solution's quality. The value of the PM is adjusted depending on the number of consecutive iterations in which the swarm has succeeded (#successes) or failed (#failures) to improve the best solution found so far, where iteration $t \to t+1$ is a success if $f(\vec{g}_{t+1}) < f(\vec{g}_t)$, a failure otherwise. The PM-success rate strategy is defined as follows:

$$\text{PM} = \begin{cases} \text{PM} \cdot 2 & \text{if \#successes} > s_c \\ \text{PM} \cdot 0.5 & \text{if \#failures} > f_c \\ \text{PM} & \text{otherwise} \end{cases}, \tag{8.16}$$

where the threshold parameters s_c and f_c are user defined.

8.3.4 Mtx Component

The options for the Mtx metaheuristic component in PSO-*X* are Mtx-random diagonal, Mtx-random linear, Mtx-exponential map, and Mtx-Euclidean rotation. The Mtx-random diagonal and Mtx-random linear options are both $d \times d$ diagonal matrices whose values are drawn from a $\mathcal{U}(0,1)$; the only difference between them is that, in Mtx-random linear, one random value is repeated d times in the matrix diagonal, whereas, in Mtx-random diagonal, the matrix contains d independently sampled values.

The Mtx-exponential map (Wilke et al. 2007) option is based on an approximation method called exponential map whereby RRMs can be constructed avoiding matrix multiplication, which is computationally expensive. Mtx-exponential map is defined as:

$$\text{Mtx-exponential map} = I + \sum_{\beta=1}^{max_\beta} \frac{1}{\beta!}\left(\frac{\alpha\pi}{180}\left(A - A^T\right)\right), \tag{8.17}$$

where I is the identity matrix, α is a scalar representing the rotation angle, and A is an $n \times n$ random matrix with uniform random numbers in $[-0.5, 0.5]$. To keep the computational complexity low we set $max_\beta = 1$.

The Mtx-Euclidean rotation (Bonyadi et al. 2014) rotates a vector in any combination of planes.[5] An Mtx-Euclidean rotation for rotating axis x_i in the direction of x_j by the angle α is given by a matrix $[r_{mn}]$ with $r_{ii} = r_{jj} = \cos\alpha$, $r_{ij} = -\sin\alpha$, $r_{ji} = \sin\alpha$, and the remaining values are set to 1 if they are on the diagonal or to zero otherwise. Since $[r_{mn}]$ is an identity matrix except for the entries at the intersections between rows i and j and columns i and j, the multiplication between $[r_{mn}]$ and $\vec{v}$ is done as follows:

$$[r_{mn}]\,\vec{v} = \begin{cases} v_k r_{ii} + v_j r_{ji} & \text{if } k = i \\ v_k r_{jj} + v_i r_{ij} & \text{if } k = j \\ v_k & \text{otherwise} \end{cases}, \tag{8.18}$$

where v_k indicates the k^{th} entry of vector $\vec{v}$. We use Mtx-Euclidean rotation$_{all}$ to indicate when Mtx-Euclidean rotation is used to rotate a vector in all possible combination of planes, and Mtx-Euclidean rotation$_{one}$ to indicate when it is used

[5] For a d-dimensional vector $\vec{u}$, there is a composition of $d(d-1)/2$ dimensional rotation matrices built up in order to rotate $\vec{u}$ in all possible combinations of planes. See (Bonyadi et al. 2014, Appendix III) for further details.

to rotate in only one plane.

The strategies to compute the rotation angle are α-constant, α-Gaussian and α-adaptive. In α-constant, the value of α is defined by the user, whereas in α-Gaussian and α-adaptive, it is obtained by sampling values from $\mathcal{N}(0, \sigma)$. The value of σ when the Gaussian distribution is used can be a user defined parameter, as in α-Gaussian, or be computed using an adaptive approach, as in α-adaptive, which is defined as follows:

$$\sigma = \frac{\zeta \times ir_t}{\sqrt{d}} + \rho, \tag{8.19}$$

where ζ and ρ are two parameters and ir_t is the number of improved particles in the last iteration divided by the population size.

The last option for the Mtx component is Mtx-Increasing group-based (Zyl and Engelbrecht 2016) that divides a random diagonal matrix into g_t groups and every element in each group has the same value, generated uniformly random. The number of groups at each iteration is computed using the following equation:

$$g_t = \frac{d-1}{t_{max}-1} \times (t-1) + 1, \tag{8.20}$$

where d is the number of problem dimensions and t_{max} is the iteration number at which the implementation stops. Note the implementation starts with $g_t = 1$ and the number increases over time until there are $g_t = d$ groups, which is equivalent to gradually transforming an Mtx-random linear component into a Mtx-random diagonal one.

8.3.5 Topology, Model of Influence and Population Components

In addition to the well-known options for the Topology component discussed in Section 8.1.1 and 8.2 and showed in Table 8.2, we implemented in PSO-*X* the Top-hierarchical and Top-time-varying options.

In Top-hierarchical (Janson and Middendorf 2005), particles are arranged in a regular tree—i.e., a tree graph with a maximum branching degree (bd) and height (h)—where they move up and down based on the quality of their $\vec{p}_t$ vector, and sets N^i contain only the particles that are in the same branch of the tree as particle i but in a higher position. The topology is updated at the end of each iteration starting from the root node, and consists of each particle comparing the quality of its $\vec{p}_t$ vector with that of its parent and switching

places when it has higher quality.

The Top-time-varying (Montes de Oca et al. 2009) is a topology that reduces its connectivity over time: it starts as a fully-connected topology and every κ iterations a number of edges is randomly removed from the graph until the topology is transformed into a ring. The value of κ, which controls the velocity at which the topology is transformed, is a multiple of the number of particles in the swarm, so that the larger the value of κ the faster the topology will be disconnected. Additionally, the number of edges to be removed follows an arithmetic regression pattern of the form $n-2, n-3, \ldots, 2$, where n is the swarm size.

The options for the Model of influence component are MoI-best-of-neighborhood, where sets I^i contains i and the local best particle in the neighborhood of i; the MoI-fully informed, where sets $I^i = N^i$; MoI-ranked fully informed, which is similar to the MoI-fully informed, but particles in I^i are ranked according to their quality so that the influence of a particle with rank r is twice the influence of a particle with rank $r-1$; and MoI-random informant, which allows particles to select a random neighbor from N^i to form set I^i.

The options for the Population component are Pop-constant, Pop-time-varying and Pop-incremental. In Pop-time-varying (Hsieh et al. 2008), there is a maximum (pop_{max}) and minimum (pop_{min}) number of particles that can be in the swarm at any given time. Particles are added or removed according to two criteria: (i) add one particle if the best solution found has not improved in the previous k consecutive iterations and the swarm size is smaller than pop_{max}; and (ii) remove the particle with lowest quality if the best solution found has improved in the previous k consecutive iterations and the swarm size is larger than pop_{min}. Whenever criterion (i) is verified, but the swarm size is equal to pop_{max}, the particle with lowest quality is removed before adding the new random particle.

In Pop-incremental (Montes de Oca et al. 2010), the implementation starts with an initial number of particles (pop_{ini}) and, at each iteration, there are ξ new particles added to the swarm until a maximum number is reached (pop_{fin}).

The initial position of newly added particles (x^{new}) can be computed using Init-random or Init-horizontal. In Init-random,

$$x^{\text{new},j} = \mathcal{U}[lb^j, ub^j],$$

where lb^j and ub^j are the lower and upper bound of the j^{th} dimension of the search space. In Init-horizontal, an horizontal learning approach is applied to

$x^{\text{new},j}$ after it has been randomly initialized in the search space:

$$x'^{\text{new},j} = \mathcal{U}[lb^j, ub^j],$$
$$x^{\text{new},j} = x'^{\text{new},j} + \mathcal{U}[0,1) \cdot (g_t^j - x'^{\text{new},j}).$$

Using a dynamic population requires that the topology is updated in order to assign newly added particles to a neighborhood or to reconnect particles that were connected to a particle that was removed. This is handled as follows:

(i) *Particles are added to a fixed topology*—the topology is extended by connecting a newly added particle with a set of neighbors randomly chosen. In Top-hierarchical, new particles are always placed at the bottom of the tree.

(ii) *Particles are added to a time-varying topology*—we assign $\hat{C}_t^i$ neighbors to every new particle, where $\hat{C}_t^i$ is the average number of neighbors that every particle in the swarm has at iteration t.

(iii) *Particles are removed*—the topology is repaired to ensure that every particle has the right number of neighbors.

Table 8.2: Available options in PSO-X for Population, Topology and Model of influence metaheuristic components

Component	Option
Topology	Top-ring Top-fully-connected Top-Von Neumann Top-random edge Top-hierarchical Top-time-varying
Model of influence	MoI-best-of-neighborhood MoI-fully informed MoI-ranked fully informed MoI-random informant
Population	Pop-constant Pop-time-varying Pop-incremental
Initialization	Init-random Init-horizontal

8.3.6 Acceleration Coefficients

The four strategies that can be used to computed the acceleration coefficients (ACs) in PSO-*X* are: AC-constant, AC-random, AC-time-varying and AC-extrapolated. In AC-random, the value of φ_{1t} and φ_{2t} is drawn from $\mathcal{U}[\varphi_{min}, \varphi_{max}]$, where $0 \leq \varphi_{min} \leq \varphi_{max} \leq 2.5$ are user selected parameters. The AC-time-varying strategy is the one proposed in (Ratnaweera et al. 2004), where φ_1 decreases from 2.5 to 0.5 and φ_2 increases from 0.5 to 2.5. In the AC-extrapolated strategy, proposed in (Arumugam et al. 2009), the value of the acceleration coefficients is a function of the iteration number and particles quality computed as follows:

$$\begin{aligned} \varphi_1 &= e^{-(t/t_{max.})} \\ \varphi_2 &= e^{(\varphi_1 \cdot \Lambda_t^i)} \end{aligned} \tag{8.21}$$

where $\Lambda_t^i = |(f(\vec{l}_t^i) - f(\vec{x}_t^i))/f(\vec{l}_t)|$ adjusts the value of φ_2 in terms of the difference between $f(\vec{x}_t^i)$ and $f(\vec{l}_t^i)$. This means that when $f(\vec{l}_t^i) \ll f(\vec{x}_t^i)$, the step size of the particle will be larger, and when $f(\vec{l}_t^i) \approxeq f(\vec{x}_t^i)$ it will be smaller.

8.3.7 Re-initialization and Velocity Clamping

The last group of components in PSO-*X* are those that have been proposed with the goal of avoiding performance issues that affect PSO, such as divergence and stagnation.

The first ones is stagnation detection (Schmitt and Wanka 2013). It is used to perturb the velocity vector of a particle when its current position is too close to the global best solution, and the velocity magnitude is not large enough to let the particle move to other parts of the search space. That is, when $||\vec{v}_t^i|| + ||\vec{g}_t - \vec{x}_t^i|| \leq \mu$, where $\mu > 0$ is a user defined threshold for the perturbation to occur. When the stagnation condition is verified, the velocity vector of the particle is randomly regenerated as follows:

$$\vec{v}_t^i = (2\vec{r} - 1) \cdot \mu,$$

where $\vec{r} \sim \mathcal{U}(0, 1]$.

The second component, particles reinitialization (García-Nieto and Alba 2011) is used to regenerate the position vector of the particles in case of early stagnation or ineffective movement is occurring. Early stagnation is considered to be affecting the implementation when the standard deviation of the $\vec{p}_t$ vectors

is lower than 0.001. In this case, each entry of the particles position vector is randomly reinitialized with probability $1/d$. The second criterion, which tries to identify when particles are moving ineffectively, consists in detecting when the overall change of $\vec{g}_t$ is lower than 10^{-8} for $10 \cdot d/\texttt{pop}$ iterations and regenerating particles positions using the following equation:

$$x_{t+1}^{i,j} = (g_t^j - x_t^{i,j})/2 \;\text{ for }\; j = 1, \ldots, d.$$

The last ones is velocity clamping (Eberhart and Kennedy 1995; Eberhart and Shi 2000) and consists in restricting the values of each dimension in the velocity vector of a particle within certain limits to prevent overly large steps. This is done using the following equation:

$$\vec{v}_{t+1}^{\,j} \begin{cases} v^{max} & \text{if } \vec{v}_{t+1}^{\,j} > v^{max} \\ -v^{max} & \text{if } \vec{v}_{t+1}^{\,j} < -v^{max} \\ \vec{v}_{t+1}^{\,j} & \text{otherwise} \end{cases}, \tag{8.22}$$

where v_{max}^j and $-v_{max}^j$ are maximum and minimum allowable value for the particle's velocity in dimension j. The value $v_{max}^j = \frac{ub^j - lb^j}{2}$ is set according to the lower lb^j and upper ub^j bounds for dimension j on the search space.

8.4 Experimental Procedure

8.4.1 Benchmark Problems

We conducted experiments on a set of 50 static benchmark continuous functions belonging to the CEC'05 and CEC'14 "Special Session on Single Objective Real-Parameter Optimization" (Liang et al. 2005; Suganthan et al. 2005), and to the Soft Computing (SOCO'10) "Test Suite on Scalability of Evolutionary Algorithms and other Metaheuristics for Large Scale Continuous Optimization Problems" (Herrera et al. 2010). A detailed description of the benchmark functions can be found in the given references and in the supplementary material (Camacho-Villalón et al. 2021) of the article.

The test set of continuous functions—Table 8.3—is composed of 12 unimodal functions (f_{1-12}), 14 multimodal functions (f_{13-26}), and 24 hybrid composition functions (f_{27-50}). With the exception of f_{41}, none of the hybrid composition

functions is separable, and the ones from f_{42-50} include also a rotation in the objective function.

8.4.2 Experimental Setup

The computational budget used with `irace` was of 50 000 executions for creating the PSO-X algorithms and of 15 000 executions for tuning the parameter of the PSO variants included in our comparison. The reason for using different budgets is that there are 58 parameters involved in the creation of the PSO-X algorithms, and only between 5 and 10 parameters in the tuning of the PSO variants. The functions employed for creating and configuring the algorithms with `irace` (i.e., the training instances) used $d = 30$, and the ones used for our experimental evaluation used $d = 50$ and $d = 100$, depending on the scalability of each function. In order to present statistically meaningful results, we perform 50 independent runs of each algorithms on each function and report the median (MED) result—to measure the quality of the solutions produced by the algorithms—and the median error (MEDerr) with respect to the best solution found by any of the algorithms.

In all cases, the algorithms were stopped after reaching $5\,000 \times d$ objective function evaluations (FEs). Both the tuning and the experiments were carried out on single core Intel Xeon E5-2680 running at 2.5GHz with 12 Mb cache size under Cluster Rocks Linux version 6.0/CentOS 6.3. The PSO-X framework was codified using `C++` and compiled with `gcc` 4.4.6.[6] The version of `irace` is 3.2.

8.5 Analysis of the Results

The analysis of the results is divided into two parts. In the first part, we analyze the performance and capabilities of six automatically generated PSO-X algorithms, named PSO-X_{all}, PSO-X_{hyb}, PSO-X_{mul}, PSO-X_{uni}, PSO-X_{cec} and PSO-X_{soco}. Each of these PSO-X algorithms has been created using a set of training instances composed of different functions. For PSO-X_{all}, we used all the fifty functions (f_{1-50}), whereas for PSO-X_{uni} we used only the unimodal functions (f_{1-12}), for PSO-X_{mul} only the multimodal ones (f_{13-26}) and for PSO-X_{hyb} only the hybrid compositions (f_{27-50}). In the case of PSO-X_{cec} and PSO-X_{soco}, we used the entire set of functions of the CEC'05 and SOCO'10 test suites, respectively.

[6]The source code of PSO-X can be downloaded from http://iridia.ulb.ac.be/supp/IridiaSupp2021-001/PSO-X.zip.

Table 8.3: Benchmark functions

$f_{\#}$	**Name**	**Search Range**	**Suite**
f_1	Shifted Sphere	[-100,100]	SOCO
f_2	Shifted Rotated High Conditioned Elliptic	[-100,100]	CEC'14
f_3	Shifted Rotated Bent Cigar	[-100,100]	CEC'14
f_4	Shifted Rotated Discus	[-100,100]	CEC'14
f_5	Shifted Schwefel 22.1	[-100,100]	SOCO
f_6	Shifted Rotated Schwefel 1.2	[-65.536,65.536]	SOCO
f_7	Shifted Scfewels12 noise in fitness	[-100,100]	CEC'05
f_8	Shifted Schwefel 2.22	[-10,10]	SOCO
f_9	Shifted Extended f_{10}	[-100,100]	SOCO
f_{10}	Shifted Bohachevsky	[-100,100]	SOCO
f_{11}	Shifted Schaffer	[-100,100]	CEC'05
f_{12}	Shchwefel 2.6 Global Optimum on Bounds	[-100,100]	CEC'05
f_{13}	Shifted Ackley	[-32,32]	SOCO
f_{14}	Shifted Rotated Ackley	[-100,100]	CEC'14
f_{15}	Shifted Rosenbrock	[-100,100]	SOCO
f_{16}	Shifted Rotated Rosenbrock	[-100,100]	CEC'14
f_{17}	Shifted Griewank	[-600,600]	SOCO
f_{18}	Shifted Rotated Griewank	[-100,100]	CEC'14
f_{19}	Shifted Rastrigin	[-100,100]	SOCO
f_{20}	Shifted Rotated Rastrigin	[-100,100]	CEC'14
f_{21}	Shifted Schwefel	[-100,100]	SOCO
f_{22}	Shifted Rotated Schwefel	[-100,100]	CEC'14
f_{23}	Shifted Rotated WeierStrass	[-100,100]	CEC'05
f_{24}	Shifted Rotated Katsuura	[-100,100]	CEC'14
f_{25}	Shifted Rotated HappyCat	[-100,100]	CEC'14
f_{26}	Shifted Rotated HGBat	[-100,100]	CEC'14
f_{27}	Hybrid Function 1 (N = 2)	[-100,100]	SOCO
f_{28}	Hybrid Function 2 (N = 2)	[-100,100]	SOCO
f_{29}	Hybrid Function 3 (N = 2)	[-5,5]	SOCO
f_{30}	Hybrid Function 4 (N = 2)	[-10,10]	SOCO
f_{31}	Hybrid Function 7 (N = 2)	[-100,100]	SOCO
f_{32}	Hybrid Function 8 (N = 2)	[-100,100]	SOCO
f_{33}	Hybrid Function 9 (N = 2)	[-5,5]	SOCO
f_{34}	Hybrid Function 10 (N = 2)	[-10,10]	SOCO
f_{35}	Hybrid Function 1 (N = 3)	[-100,100]	CEC'14
f_{36}	Hybrid Function 2 (N = 3)	[-100,100]	CEC'14
f_{37}	Hybrid Function 3 (N = 4)	[-100,100]	CEC'14
f_{38}	Hybrid Function 4 (N = 4)	[-100,100]	CEC'14
f_{39}	Hybrid Function 5 (N = 5)	[-100,100]	CEC'14
f_{40}	Hybrid Function 6 (N = 5)	[-100,100]	CEC'14
f_{41}	Hybrid Composition Function	[-5,5]	CEC'05
f_{42}	Rotated Hybrid Composition Function	[-5,5]	CEC'05
f_{43}	Rotated H. Composition F. with Noise in Fitness	[-5,5]	CEC'05
f_{44}	Rotated Hybrid Composition F.	[-5,5]	CEC'05
f_{45}	Rotated H. Composition F. with a Narrow Basin for the Global Opt.	[-5,5]	CEC'05
f_{46}	Rotated H. Comp. F. with the Gbl. Opt. On the Bounds	[-5,5]	CEC'05
f_{47}	Rotated Hybrid Composition Function	[-5,5]	CEC'05
f_{48}	Rotated H. Comp. F. with High Condition Num. Matrix	[-5,5]	CEC'05
f_{49}	Non-Continuous Rotated Hybrid Composition Function	[-5,5]	CEC'05
f_{50}	Rotated Hybrid Composition Function	[-5,5]	CEC'05

Unlike the SOCO'10 test suite, the CEC'05 competition set includes many rotated objective functions and more complex hybrid compositions. The idea of using different training instances is to try to identify the metaheuristic components that result in higher performance when tackling functions of different classes.

In the second part, we compare the performance of our automatically generated PSO-*X* algorithms with ten well-known variants of PSO. We used two versions of each PSO variant: one whose parameters were tuned with `irace` (indicated by "*tnd*") and the other that uses the default parameter settings proposed by the original authors (indicated by "*dft*"). The PSO variants included in our comparison are the following:

1. Enhanced rotation invariant PSO (Bonyadi et al. 2014) (ERiPSO)—a variant that uses the AC-random strategy, Mtx-Euclidean rotation and α-adaptive.

2. Fully informed PSO (Mendes et al. 2004) (FiPSO)—a traditional PSO variant that uses the constriction coefficient velocity update rule[7] (CCVUR) and the Mol-fully informed.

3. Frankenstein's PSO (Montes de Oca et al. 2009) (FraPSO)—a PSO variant that uses Top-time-varying, Mol-fully informed and $\omega_1 =$ linear decreasing.

4. Gaussian "bare-bones" PSO (Kennedy 2003) (GauPSO)—a variant that uses the DNPP-Gaussian option of the DNPP-additive stochastic as the only mechanism to update particles positions.

5. Hierarchical PSO (Janson and Middendorf 2005) (HiePSO)—a variant based on Top-hierarchical that can be implemented using either $\omega_1 =$ linear decreasing or $\omega_1 =$ linear increasing.

6. Incremental PSO (Montes de Oca et al. 2010) (IncPSO)—a variant of PSO that uses the CCVUR and Pop-incremental with Init-horizontal.

7. Locally convergent rotation invariant PSO (Bonyadi and Michalewicz 2014) (LcRPSO)—a more recent variant of PSO in which the $\text{Pert}_{\text{info}}$-Gaussian component is used together with the PM-Euclidean distance strategy, Mtx-random linear and the AC-random strategy.

[7] This rule is define as: $\vec{v}_{t+1}^{\,i} = \chi(\vec{v}_t^{\,i} + \varphi_1 U_{1t}^i(\vec{p}_t^{\,i} - \vec{x}_t^{\,i}) + \varphi_2 U_{2t}^i(\vec{l}_t^{\,i} - \vec{x}_t^{\,i}))$, where $\chi = 0.7298$ is called constriction coefficient (Clerc and Kennedy 2002). It can obtained from Equation 8.4 by setting $\omega_1 = \omega_2 = 0.7298$ and using the DNPP-rectangular option.

8. Restart PSO (García-Nieto and Alba 2011) (ResPSO)—a variant of StaPSO using velocity clamping and particles reinitialization.

9. Standard PSO (Shi and Eberhart 1998) (StaPSO)—the PSO algorithm described in Section 8.1.1 that uses Equation 8.1, 8.2 and 8.3.

10. Standard PSO 2011 (SPSO11)—a variant of StaPSO that uses the DNPP-spherical option.

In Table 8.4, we show the parameter configuration of the versions that we used in the comparison. Note that, with the goal of simplifying their description, we have only mentioned the components that are different in these algorithms from those in StaPSO. This means that, unless specified otherwise, we assumed that the following components and parameters setting are used in their implementation: Pop-constant, Top-fully-connected with Mol-best-of-neighborhood, DNPP-rectangular with Mtx-random diagonal and $\text{Pert}_{\text{info}} = \text{Pert}_{\text{rand}} =$ none, $\omega_1 =$ constant, $\omega_2 = 1.0$, $\omega_3 = 0$ and AC-constant.

8.5.1 Comparison of Automatically Generated PSO Algorithms

The metaheuristic components in the automatically generated PSO-X algorithms are listed below, and their configuration is given in Table 8.5.

- PSO-X_{all}: Pop-incremental with Init-random, Top-fully-connected with Mol-best-of-neighborhood, DNPP-rectangular with $\text{Pert}_{\text{info}}$-Lévy and PM-success rate, Mtx-random diagonal and velocity clamping.

- PSO-X_{hyb}: Pop-constant, Top-Von Neumann with Mol-best-of-neighborhood, DNPP-rectangular with $\text{Pert}_{\text{info}}$-Lévy and PM-success rate, Mtx-random diagonal and velocity clamping.

- PSO-X_{mul}: Pop-incremental with Init-horizontal, Top-time-varying with Mol-best-of-neighborhood, DNPP-rectangular with $\text{Pert}_{\text{info}}$-Lévy and PM-success rate, Mtx-random linear and stagnation detection.

- PSO-X_{uni}: Pop-incremental with Init-random, Top-fully-connected with Mol-best-of-neighborhood, DNPP-rectangular with $\text{Pert}_{\text{info}}$-Lévy and PM-success rate, Mtx-random diagonal and velocity clamping.

- PSO-X_{cec}: Pop-constant, Top-Von Neumann with Mol-best-of-neighborhood, DNPP-rectangular with $\text{Pert}_{\text{info}}$-Lévy and PM-success rate, $\text{Pert}_{\text{rand}}$-noisy with PM-success rate and Mtx-random diagonal.

Table 8.4: Parameter settings of the ten PSO variants included in the experimental comparison.

Algorithm	Settings
ERiPSO$_{dft}$	pop $= 20$, $\omega_1 = 0.7213475$, AC-random, $\varphi_{1min} = 0$, $\varphi_{1max} = 2.05$, $\varphi_{2min} = 0$, $\varphi_{2max} = 2.05$, Mtx-Euclidean rotation$_{all}$ with α-adaptive and $\zeta = 30$ and $\rho = 0.01$.
FiPSO$_{tnd}$	Top-ring, pop $= 20$, $\omega_1 = \omega_2 = 0.729843788$, Mtx-random diagonal, $\varphi_1 = 2.1864$ and $\varphi_2 = 2.3156$.
FraPSO$_{dft}$	$\kappa = 60$, pop $= 60$, $\omega_1 =$ linear decreasing, $\omega_{1min} = 0.4$, $\omega_{1max} = 0.9$, $t_{schd} = 600$, $\varphi_1 = 2.0$ and $\varphi_2 = 2.0$.
GauPSO$_{tnd}$	Top-time-varying with Mol-random informant, $\kappa = 150$ pop $= 30$, $\omega_1 = 0$ and DNPP-additive stochastic DNPP-Gaussian.
HiePSO$_{tnd}$	$bd = 2$, pop $= 114$, $\omega_1 =$ linear increasing, $\omega_{1min} = 0.3284$, $\omega_{1max} = 0.8791$, $\varphi_1 = 2.1105$ and $\varphi_2 = 1.0349$.
IncPSO$_{tnd}$	Top-time-varying, $\kappa = 2360$, Init-horizontal, pop$_{ini} = 5$, pop$_{fin} = 295$, $\xi = 10$, $\omega_1 = \omega_2 = 0.729843788$, $\varphi_1 = 1.9226$ and $\varphi_2 = 1.0582$.
LcRPSO$_{dft}$	pop $= d$, $\omega_1 = 0.7298$, AC-random, $\varphi_{1min} = 0$, $\varphi_{1max} = 1.4962$, $\varphi_{2min} = 0$, $\varphi_{2max} = 1.4962$, Mtx-random linear, Pert$_{\text{info}}$-Gaussian with PM-Euclidean distance and $\epsilon = 0.46461/d^{0.79}$.
ResPSO$_{tnd}$	Top-ring with Mol-fully informed, pop $= 10$, $\omega_1 =$ linear decreasing, $\omega_{1min} = 0.2062$, $\omega_{1max} = 0.6446$, $\varphi_1 = 1.5014$, $\varphi_2 = 2.2955$.
SPSO11$_{tnd}$	Top-time-varying with $\kappa = 1085$, pop $= 155$, $\omega_1 = 0.6482$, $\varphi_1 = 2.2776$, $\varphi_2 = 2.1222$.
StaPSO$_{tnd}$	Top-Von Neumann, pop $= 34$, $\omega_1 = 0.6615$, $\varphi_1 = 2.3706$, $\varphi_2 = 0.8914$.

[*] As reminder for the reader, ζ and ρ are parameters of α-adaptive; σ is a parameter of α-Gaussian; κ is a parameter of Top-time-varying; t_{schd} is a parameter of $\omega_1 =$ linear decreasing; bd is a parameter of Top-hierarchical; ξ is a parameter of Pop-incremental; and ϵ is a parameter of PM-Euclidean distance.

- PSO-X_{soco}: Pop-constant, Top-ring with Mol-ranked fully informed, DNPP-rectangular with Pert$_{\text{info}}$-Gaussian and PM-success rate, Mtx-random diagonal and velocity clamping.

In Table 8.6, we report the median of the results obtained by the algorithms on each function. At the bottom of the table, we show the number of times each algorithm obtained the best result among the six ("Wins"), the average median value ("Av.MED"), the average ranking of the algorithm across all 50 functions ("Av.Ranking"), and whether the overall performance of any of the compared algorithm was significantly worse ("+") or equal ("≈") than the best ranked

Table 8.5: Parameter settings of the six automatically created PSO-X algorithms.

Algorithm	Settings
PSO-X_{all}	$\text{pop}_{ini} = 4$, $\text{pop}_{fin} = 20$, $\xi = 8$, $\omega_1 =$ convergence-based, $a = 0.7192$, $b = 0.9051$, $\omega_2 =$ random, AC-constant, $\varphi_1 = 1.7067$, $\varphi_2 = 2.2144$, PM $= 0.438$, $s_c = 11$ and $f_c = 40$.
PSO-X_{hyb}	pop $= 41$, $\omega_1 =$ adaptive based onvelocity, $\omega_{1min} = 0.119$, $\omega_{1max} = 0.1378$, $\lambda = 0.608$, $\omega_2 = 1.0$ AC-random, $\varphi_{1min} = 1.0429$, $\varphi_{1max} = 2.1653$, $\varphi_{2min} = 1.0429$, $\varphi_{2max} = 2.3275$, PM $= 0.5333$, $s_c = 28$ and $f_c = 42$.
PSO-X_{mul}	$\kappa = 300$, $\text{pop}_{ini} = 3$, $\text{pop}_{fin} = 50$, $\xi = 2$, $\omega_1 =$ success-based, $\omega_{1min} = 0.4$, $\omega_{1max} = 0.9$, AC-constant, $\varphi_1 = 0.92$, $\varphi_2 = 1.6577$, PM $= 0.5114$, $s_c = 2$ and $f_c = 33$.
PSO-X_{uni}	$\text{pop}_{ini} = 10$, $\text{pop}_{fin} = 58$, $\xi = 3$, $\omega_1 =$ adaptive based onvelocity, $\omega_{1min} = 0.3531$, $\omega_{1max} = 0.7095$, $\lambda = 0.4832$, $\omega_2 = \omega_1$, AC-random, $\varphi_{1min} = 1.4217$, $\varphi_{1max} = 2.051$, $\varphi_{2min} = 0.8626$, $\varphi_{2max} = 1.4609$, PM $= 0.9865$, $s_c = 38$ and $f_c = 11$.
PSO-X_{cec}	pop $= 42$, $\omega_1 =$ self-regulating, $\omega_{1min} = 0.1673$, $\omega_{1max} = 0.2317$, $\eta = 0.2468$, $\omega_2 =$ random, $\omega_3 =$ random, AC-random, $\varphi_{1min} = 1.8684$, $\varphi_{1max} = 1.9233$, $\varphi_{2min} = 0.2802$, $\varphi_{2max} = 1.5143$, $\text{PM}_1 = 0.4837$, $s_{c1} = 29$, $f_{c1} = 45$, $\text{PM}_2 = 0.8139$, $s_{c2} = 30$ and $f_{c2} = 43$.
PSO-X_{soco}	pop $= 19$, $\omega_1 =$ adaptive based onvelocity, $\omega_{1min} = 0.6564$, $\omega_{1max} = 0.8201$, $\lambda = 0.2959$, AC-constant, $\varphi_1 = 0.7542$, $\varphi_2 = 1.9235$, $\text{PM}_{t=0} = 0.8907$, $s_c = 22$ and $f_c = 49$.

* As reminder for the reader, ξ is a parameter of Pop-incremental; a and b are parameters of $\omega_1 =$ convergence-based, λ of $\omega_1 =$ adaptive based onvelocity, and η of $\omega_1 =$ self-regulating; s_c and f_c are parameters of PM-success rate; and κ is a parameter of Top-time-varying;.

algorithm according to a Wilcoxon's rank-sum test at 0.95 confidence interval with Bonferroni's correction. PSO-X_{all} was the algorithm that ranked best of the six followed by PSO-X_{cec} and PSO-X_{hyb}, while PSO-X_{soco} was the one that returned the best median result in the higher number of cases. The symbol "+" next to some of the median values in Table 8.6 indicates the cases where we found a statistical difference function-wise in favor of PSO-X_{all} according also to a Wilcoxon-Bonferroni test with $\alpha = 0.05$. PSO-X_{uni}, which ranked last of the six, was the algorithm that performed statistically worse than PSO-X_{all} in most functions (26 out of 50 functions), while PSO-X_{mul}, PSO-X_{cec}, PSO-X_{hyb}, and PSO-X_{soco} were worse in 23, 19, 17, and 14 functions, respectively. In the following we examine the performance of the six PSO-X algorithms across the different function classes in our benchmark set, focusing on those that are

specific to a function class, and the effect of their algorithm differences in their performance.

Comparison of the PSO-*X* Algorithms on Specific Function Classes

In order to know whether our PSO-*X* algorithms are able to obtain better results in specific function classes, we analyze their performance according to the average ranking (Av.Ranking) they obtained in the unimodal (f_{1-12}), multimodal (f_{13-26}), hybrid composition (f_{27-50}) and rotated ($f_{\text{rotated}} = f_{2-4,6,14,16,18,20,22-26,42-50}$) functions. The Av.Ranking gives us an indication of how good or bad is the performance of an algorithm across the different classes based on the result of the winner of each function. In Table 8.7, we present this information together with the algorithms average median error (Av.MEDerr). In our analysis, we pay particular attention to the results of PSO-X_{uni}, PSO-X_{mul}, PSO-X_{hyb} and PSO-X_{cec}, that are the algorithms we would expect to obtain better results because of the functions used for creating them.

As shown in Table 8.7, according to the median solution quality, the performance of the algorithms is weakly correlated with the class of functions used with `irace`. Although PSO-X_{mul} ranked first in its function class of specialization, PSO-X_{uni} was outperformed by all the algorithms in the unimodal functions, PSO-X_{hyb} was outperformed by PSO-X_{all} in the hybrid compositions, and PSO-X_{cec} was outperformed by PSO-X_{hyb} and PSO-X_{mul} in the rotated functions. An analysis of the results using the average median error of the algorithms shows similar results, although, in this case, the performance of PSO-X_{uni} and PSO-X_{mul} was weakly correlated to the class of functions used in their training sets.

There are a few possible reasons why the use of different sets of functions did not have a stronger effect on the performance of our algorithms. The first one is the way in which we separated the functions, that captures some features of the functions, but neglects others, such as separability, noise, and different combination of objective functions transformations.[8] Another possible reason is the presence of slightly overfitted models during the creation of these algorithms with `irace`. The effect of overfitting can be observed more clearly for PSO-X_{uni} than for the rest of algorithms. For a number of functions (e.g., $f_{6,12}$) the median solution obtained by PSO-X_{uni} was significantly better than that of the other

[8]Note, for example, that there are rotated functions in the training set of the six PSO-*X* algorithms, except for PSO-X_{soco} that includes only translations.

algorithms, which contributes to lower the value of the Av.MED and Av.MEDerr metrics, but not to improve its ranking in its respective classes of specialization. Among the possible causes for the overfitting are the use of training sets with different number of instances (PSO-X_{uni} has 12 instances, while the best ranked algorithm, PSO-X_{all}, has 50) and of a exceedingly large computational budget used with `irace`.

PSO-*X* Algorithm Differences

The first thing to note about the design of the six PSO-*X* algorithms is that, despite they were created using different sets of functions, they all share the same core components, i.e., DNPP-rectangular with $\text{Pert}_{\text{info}}$-Lévy or $\text{Pert}_{\text{info}}$-Gaussian. This combination of components, as we discusses in Section 8.3.2, has the ability of making the implementation rotation invariant, which is an important characteristic given that 22 out of the 50 functions in our benchmark test set have a rotation in their objective function. In all cases, the strategy to control the perturbation magnitude (PM) was PM-success rate and, with the exception of PSO-X_{uni}, they all have a parameter setting where f_c is larger than s_c. This setting allows to decrease rapidly the PM when particles have been constantly improving the global best solution, but makes harder to switch back to a larger PM if the algorithm happens to stagnate. In this sense, PSO-X_{all}, PSO-X_{hyb}, PSO-X_{mul}, PSO-X_{cec} and PSO-X_{soco} are biased towards exploitation, and PSO-X_{uni} towards exploration.

Although PSO-X_{hyb} and PSO-X_{cec} obtained similar results in most functions and ranked almost the same across the whole benchmark set, the performance of PSO-X_{cec} was better in the CEC'05 hybrid compositions (f_{41-50}), and worse in functions f_{27}, f_{30}, f_{31} and f_{34} that belong to the SOCO'10 test suite. Based on the components and parameter setting in PSO-X_{hyb} and PSO-X_{cec}, this difference can be attributed to the $\text{Pert}_{\text{rand}}$ component that is present only in PSO-X_{cec}. The $\text{Pert}_{\text{rand}}$ component was advantageous for PSO-X_{cec} to tackle the more complex search spaces of the CEC'05 hybrid compositions, where the algorithm performed its best, but affected its solutions quality in most of the SOCO'10 test suite hybrid compositions. Another interesting comparison can be done between PSO-X_{all} (ranked first) and PSO-X_{uni} (ranked last). These two algorithms have the exact same components and differ only in the population size, which is roughly three times larger in PSO-X_{uni} compared to PSO-X_{all}; parameter ω_1, which is equal to ω_2 in PSO-X_{uni} and random in PSO-X_{all}; and

parameters f_c and s_c, whose value is inverted in PSO-X_{uni} compare to PSO-X_{all} (see Table 8.4). Data from Table 8.6 shows that the configuration of PSO-X_{uni} is quite performing to tackle functions with large plateaus and quite regular landscapes, such as Elliptic (f_2), Schwefel (f_6, f_8, and f_{12}) or Rosenbrock (f_{15} and f_{16}), where PSO-X_{uni} was the best performing of the six. However, when PSO-X_{uni} faced less regular and multimodal landscapes, its performance declined significantly. Given the large number of parameters in PSO-X compared to most PSO variants in the literature, framework users could be interested in obtaining information about the sampling distribution of the parameters and the way in which they interact with each other. We present this information in Section B.2.2 of Appendix B, where we have included a number of plots and charts generated using the data gathered from the configuration process with `irace`.

Table 8.6: Median results of the PSO-X algorithms in f_{1-50} with $d = 50$.

f#	PSO-X_{all}	PSO-X_{uni}	PSO-X_{mul}	PSO-X_{hyb}	PSO-X_{cec}	PSO-X_{soco}
f_1	**0.00E+00**	**0.00E+00**	9.90E-09$^+$	**0.00E+00**	**0.00E+00**	**0.00E+00**
f_2	1.78E+06	**1.18E+06**	3.50E+06$^+$	2.89E+06$^+$	3.01E+06$^+$	8.33E+06$^+$
f_3	2.10E+03	6.49E+03	2.09E+03	**1.78E+03**	2.37E+03	3.07E+03
f_4	**1.46E+04**	2.31E+04$^+$	3.91E+04$^+$	1.49E+04	2.35E+04$^+$	1.65E+04
f_5	**3.76E-09**	1.05E-03$^+$	2.49E-04$^+$	5.69E-03$^+$	2.96E-07$^+$	1.55E-02$^+$
f_6	5.54E-04	**5.25E-06**	3.17E+01$^+$	2.64E+01$^+$	2.73E+00$^+$	5.64E+01$^+$
f_7	1.96E+04	4.15E+04$^+$	1.83E+04	1.45E+04	9.10E+03	**3.80E+03**
f_8	**0.00E+00**	**0.00E+00**	7.04E-04$^+$	**0.00E+00**	**0.00E+00**	**0.00E+00**
f_9	2.76E+01	1.89E+02$^+$	1.84E+02$^+$	7.98E+01$^+$	3.49E+01	**5.36E-03**
f_{10}	**0.00E+00**	1.05E+00$^+$	4.89E-07$^+$	**0.00E+00**	**0.00E+00**	**0.00E+00**
f_{11}	2.86E+01	1.95E+02$^+$	1.77E+02$^+$	7.94E+01$^+$	4.64E+01	**1.07E-02**
f_{12}	9.86E-05	**1.86E-05**	2.65E+01$^+$	6.27E-02$^+$	1.87E-01$^+$	3.03E-02$^+$
f_{13}	**-1.44E-16**	**-1.44E-16**	5.18E-05$^+$	**-1.44E-16**	**-1.44E-16**	**-1.44E-16**
f_{14}	2.11E+01	**2.00E+01**	**2.00E+01**	**2.00E+01**	**2.00E+01**	2.12E+01$^+$
f_{15}	4.60E+01	**3.35E+01**	4.64E+01	4.42E+01	4.39E+01	4.19E+01
f_{16}	4.78E+01	**4.70E+01**	**4.70E+01**	**4.70E+01**	**4.70E+01**	**4.70E+01**
f_{17}	7.40E-03	1.63E-19	2.03E-09	1.63E-19	1.63E-19	**5.42E-20**
f_{18}	**1.63E-19**	3.79E-19	1.10E-06	1.05E-12	1.20E-13	1.08E-12
f_{19}	**1.24E+01**	1.40E+02$^+$	3.11E+01$^+$	1.14E+02$^+$	8.71E+01$^+$	1.41E+02$^+$
f_{20}	2.63E+02	2.43E+02	2.03E+02	**1.95E+02**	2.18E+02	2.39E+02
f_{21}	2.44E+04	2.56E+04$^+$	**2.34E+04**	2.54E+04$^+$	2.51E+04$^+$	2.86E+04$^+$
f_{22}	2.86E+04	2.90E+04	**2.79E+04**	2.82E+04	2.81E+04	3.47E+04$^+$
f_{23}	3.60E+01	3.37E+01	2.78E+01	3.52E+01	3.74E+01	**2.29E+01**
f_{24}	2.63E-01	7.06E-01$^+$	**6.10E-02**	7.48E-01$^+$	6.20E-01$^+$	3.53E+00$^+$

* The symbol $^+$ that appears next to the median value indicates the cases where there is a statistical difference in favor of PSO-X_{all}.

Table 8.6 Continued.

f#	PSO-X_{all}	PSO-X_{uni}	PSO-X_{mul}	PSO-X_{hyb}	PSO-X_{cec}	PSO-X_{soco}
f_{25}	6.60E-01	6.20E-01	**4.10E-01**	4.67E-01	4.48E-01	4.37E-01
f_{26}	3.40E-01	7.77E-01^{+}	3.22E-01	**2.92E-01**	2.94E-01	3.40E-01
f_{27}	1.89E+01	1.21E+02^{+}	3.97E+01^{+}	**1.34E-09**	3.08E+01^{+}	2.01E+01
f_{28}	**5.52E+01**	1.47E+02^{+}	1.06E+02^{+}	1.18E+02^{+}	1.10E+02^{+}	6.50E+01
f_{29}	**1.40E+01**	7.97E+01^{+}	3.39E+01^{+}	6.52E+01^{+}	5.99E+01^{+}	6.18E+01^{+}
f_{30}	6.36E-13	1.63E-07^{+}	6.61E-04^{+}	**0.00E+00**	1.33E-07^{+}	**0.00E+00**
f_{31}	**2.80E+01**	2.49E+02^{+}	1.09E+02^{+}	6.28E+01^{+}	1.41E+02^{+}	4.02E+01
f_{32}	1.79E+02	3.89E+02^{+}	2.68E+02	2.90E+02	2.87E+02	**7.17E+01**
f_{33}	1.92E+01	7.79E+01^{+}	4.26E+01^{+}	6.32E+01^{+}	6.41E+01^{+}	**9.75E+00**
f_{34}	3.81E-18	1.34E-07^{+}	4.20E-04^{+}	2.50E-19	5.15E-08^{+}	**0.00E+00**
f_{35}	1.27E+04	**1.23E+04**	**1.23E+04**	**1.23E+04**	**1.23E+04**	**1.23E+04**
f_{36}	1.49E+05	2.61E+05^{+}	**1.22E+05**	1.52E+05	1.48E+05	3.32E+06^{+}
f_{37}	**3.90E+01**	4.77E+01^{+}	4.05E+01	5.35E+01^{+}	5.23E+01^{+}	5.99E+01^{+}
f_{38}	**1.65E+05**	3.10E+05^{+}	2.14E+05	2.51E+05^{+}	2.70E+05^{+}	4.61E+05^{+}
f_{39}	5.28E+00	5.49E+00	**4.24E+00**	4.50E+00	4.30E+00	5.43E+00
f_{40}	1.08E+01	1.74E+01^{+}	1.61E+01^{+}	1.44E+01^{+}	1.28E+01^{+}	**9.58E+00**
f_{41}	3.46E+02	4.00E+02^{+}	3.37E+02	2.98E+02	**2.24E+02**	3.51E+02
f_{42}	2.00E+02	2.23E+02	1.20E+02	1.09E+02	**9.44E+01**	2.77E+02
f_{43}	**2.25E+02**	3.16E+02	3.01E+02	2.38E+02	2.35E+02	3.09E+02
f_{44}	9.55E+02	9.34E+02	**9.25E+02**	**9.25E+02**	9.27E+02	9.29E+02
f_{45}	9.56E+02	9.34E+02	**9.25E+02**	9.28E+02	**9.25E+02**	9.29E+02
f_{46}	9.59E+02	9.33E+02	**9.25E+02**	9.28E+02	**9.25E+02**	9.29E+02
f_{47}	**8.02E+02**	1.02E+03^{+}	1.01E+03^{+}	1.02E+03^{+}	1.02E+03^{+}	1.01E+03^{+}
f_{48}	9.68E+02	9.58E+02	9.34E+02	**9.09E+02**	9.18E+02	9.22E+02
f_{49}	**9.76E+02**	1.02E+03^{+}	1.02E+03^{+}	1.02E+03	1.02E+03	1.01E+03
f_{50}	9.42E+02	9.84E+02^{+}	1.11E+03^{+}	9.57E+02	9.85E+02	**9.38E+02**

* The symbol $^{+}$ that appears next to the median value indicates the cases where there is a statistical difference in favor of PSO-X_{all}.

8.5.2 Comparison with Other PSO Algorithms

We also compared the algorithms generated using PSO-X with ten traditional and recently proposed PSO variants. As mentioned before, for each algorithm we collected data using both a default (*dft*) version—that uses the parameter settings proposed by the authors—and a tuned (*tnd*) version—whose parameters were configured with `irace`. Based on a Wilcoxon-Bonferroni test at $\alpha = 0.05$, we selected the best performing of the two versions of each algorithm. However, since the computed p-values were larger than 0.05 for ERiPSO, FraPSO and

Table 8.7: Average ranking (Av.Ranking) and average median error (Av.MEDerr) obtained by the PSO-X algorithms in f_{1-12} (unimodal), f_{13-26} (multimodal), f_{27-50} (hybrid) and f_{rotated} with $d = 50$.

f#		PSO-X_{all}	PSO-X_{uni}	PSO-X_{mul}	PSO-X_{hyb}	PSO-X_{cec}	PSO-X_{soco}
f_{1-12}	Av.Ranking	**2.83**	4.25	4.83	3.67	3.75	3.83
	Av.MEDerr	1.30E+05	**8.21E+04**	2.75E+05	2.22E+05	2.32E+05	6.75E+05
f_{13-26}	Av.Ranking	3.27	4.54	**3.15**	3.62	3.54	3.69
	Av.MEDerr	2.63E+02	3.75E+02	**1.45E+02**	3.04E+02	2.79E+02	9.55E+02
f_{27-50}	Av.Ranking	**1.9**	4.58	3.1	2.58	4.66	4.2
	Av.MEDerr	**6.04E+03**	1.63E+04	6.89E+03	9.58E+03	1.02E+04	1.45E+05
f_{rotated}	Av.Ranking	3.91	4.36	3.05	**2.77**	3.14	4.09
	Av.MEDerr	7.01E+04	**4.31E+04**	1.49E+05	1.20E+05	1.26E+05	3.68E+05
f_{1-50}	Av.Ranking	**3.24**	4.68	3.54	3.42	3.38	3.68
	Av.MEDerr	3.42E+04	**2.80E+04**	6.94E+04	5.81E+04	6.09E+04	2.34E+05

SPSO11, we selected the version that obtained the lower median value across the 50 functions. In Table 8.8, we show the median of the 50 runs executed by each algorithm for each function and, in Table 8.9, we show the mean ranking obtained by each algorithm according to the different classes in which we separated the functions in the benchmark set. To complement the information given in the tables, in Section B.4 of Appendix B, we present the distribution of the results obtained by the 16 compared algorithms using box plots.

In terms of the median solution quality, except for PSO-X_{uni}, the performance of the automatically generated PSO-X algorithms was better than any of the PSO variants in our comparison. PSO-X_{cec} obtained the best ranking followed by PSO-X_{mul} and PSO-X_{hyb}, and it was also the algorithm that returned the best median value in most functions. Regarding the performance of the algorithms on specific problems classes, PSO-X_{cec} obtained the best ranking according to the Av.MED result in the unimodal and rotated functions, PSO-X_{mul} the best one in the multimodal functions, and PSO-X_{all} the best one in the hybrid functions; whereas the algorithms that obtained the lower Av.MEDerr were LcRPSO$_{dft}$ in the unimodal and rotated functions, and PSO-X_{mul} in the multimodal and hybrid composition functions. To put these results in context, in Table 8.9, we have used boxes to highlight the results of the PSO variants whose ranking was equally good, or better, than any of the PSO-X algorithms.

Table 8.8: Median results returned by the six automatically generated PSO-X algorithms and ten other PSO variants in f_{1-40} with $d = 100$ and f_{41-50} with $d = 50$.

f#	PSO-X_{all}	PSO-X_{uni}	PSO-X_{mul}	PSO-X_{hyb}	PSO-X_{soco}	PSO-X_{cec}	GauPSO$_{tnd}$	ResPSO$_{tnd}$	ERiPSO$_{dft}$	FinPSO$_{tnd}$	FraPSO$_{dft}$	LcRPSO$_{dft}$	HiePSO$_{tnd}$	IncPSO$_{tnd}$	StaPSO$_{tnd}$	SPSO11$_{tnd}$
f_1	**0.00E+00**	**0.00E+00**$^+$	1.45E-08$^+$	**0.00E+00**	1.19E-28$^+$	**0.00E+00**	1.55E+03$^+$	3.00E-24$^+$	5.14E-25$^+$	**0.00E+00**	2.07E+03$^+$	3.26E-27$^+$	1.01E+01$^+$	1.16E-12$^+$	6.69E-28$^+$	2.93E+00$^+$
f_2	1.92E+07$^+$	4.44E+06	1.73E+07$^+$	1.18E+07$^+$	4.05E+07$^+$	8.82E+06	5.16E+08$^+$	9.84E+06	5.55E+06	7.69E+07$^+$	4.36E+07$^+$	**2.87E+06**	2.63E+08$^+$	1.47E+07$^+$	2.59E+07$^+$	2.28E+07$^+$
f_3	**2.83E+03**	4.85E+03	4.86E+03	3.96E+03	3.46E+03	3.00E+03	2.80E+09$^+$	5.53E+03	3.51E+03	4.88E+03	1.03E+04$^+$	3.58E+03	4.58E+07$^+$	5.38E+03	2.92E+03	4.75E+06$^+$
f_4	1.06E+04	7.67E+03	1.57E+04	1.02E+04	7.78E+03	1.98E+04	2.01E+05$^+$	1.87E+04	**9.95E+02**	7.61E+03	2.78E+04	3.82E+03	1.72E+05$^+$	2.86E+04	1.78E+04	2.01E+05$^+$
f_5	6.61E-03	4.19E+00$^+$	1.87E-02	2.22E+00$^+$	4.44E+00$^+$	**3.76E-03**	7.81E+01$^+$	6.93E+01$^+$	5.91E+01$^+$	3.41E+01$^+$	1.65E+00$^+$	5.16E+01$^+$	3.88E+01$^+$	5.58E+01$^+$	3.33E+01$^+$	6.12E+01$^+$
f_6	5.71E+01	1.97E-01	6.61E+02$^+$	1.08E+03$^+$	5.35E+03$^+$	2.94E+02	9.76E+04$^+$	2.89E+03$^+$	8.14E-02	1.38E+04$^+$	6.65E+03$^+$	**5.74E-04**	2.90E+04$^+$	2.49E+03$^+$	1.13E+04$^+$	1.08E+03$^+$
f_7	1.02E+05$^+$	1.87E+05$^+$	7.37E+04$^+$	9.49E+04$^+$	3.59E+04	4.41E+04	3.62E+05$^+$	5.26E+04	2.84E+04	6.95E+04$^+$	4.90E+04	2.38E+04	1.09E+05$^+$	**2.10E+04**	1.36E+05$^+$	6.00E+04$^+$
f_8	**0.00E+00**$^+$	**0.00E+00**	1.91E-03$^+$	**0.00E+00**	**0.00E+00**	**0.00E+00**	4.57E+01$^+$	1.09E-11$^+$	3.07E+02$^+$	**0.00E+00**	**0.00E+00**	3.37E+01$^+$	1.42E+00$^+$	1.35E-10$^+$	1.33E-15$^+$	1.62E+01$^+$
f_9	4.60E+02$^+$	5.57E+02$^+$	3.67E+02	3.76E+02	**9.52E-01**	3.50E+02	7.79E+02$^+$	9.12E+01	6.66E+02$^+$	3.31E+00	5.61E+02$^+$	6.10E+02$^+$	2.56E+02	1.55E+02	3.71E+02	6.31E+02$^+$
f_{10}	**0.00E+00**	3.15E+00$^+$	1.14E-06$^+$	8.23E-33$^+$	1.46E+00$^+$	**0.00E+00**	1.47E+01$^+$	4.20E+00$^+$	6.30E+01$^+$	5.90E+00$^+$	**0.00E+00**	6.13E+01$^+$	2.02E+01$^+$	1.78E+01$^+$	3.15E+00$^+$	6.49E+01$^+$
f_{11}	4.49E+02	5.46E+02$^+$	3.79E+02	3.83E+02	**1.03E+00**	3.86E+02	7.80E+02$^+$	1.02E+02	6.74E+02$^+$	2.41E+00	5.76E+02$^+$	6.27E+02$^+$	2.60E+02	1.84E+02	3.75E+02	6.28E+02$^+$
f_{12}	**4.38E+00**	3.29E+01	3.57E+02$^+$	1.29E+02$^+$	1.26E+01	3.68E+01	3.30E+04$^+$	4.59E+04$^+$	1.98E+04$^+$	5.63E+02$^+$	2.13E+02$^+$	7.44E+03$^+$	1.66E+04$^+$	1.81E+04$^+$	1.60E+03$^+$	3.22E+04$^+$
f_{13}	6.21E+00$^+$	3.32E-15$^+$	4.70E-05$^+$	**-1.44E-16**	1.39E-15$^+$	**-1.44E-16**	1.83E+01$^+$	1.12E+00$^+$	1.94E+01$^+$	**-1.44E-16**	1.89E+01$^+$	7.76E+00$^+$	3.53E+00$^+$	1.32E+00$^+$	3.22E-15$^+$	2.01E+01$^+$
f_{14}	2.13E+01$^+$	**2.00E+01**$^+$	**2.00E+01**$^+$	**2.00E+01**	2.13E+01$^+$	**2.00E+01**	2.13E+01$^+$	2.13E+01$^+$	2.03E+01$^+$	2.13E+01$^+$	2.13E+01$^+$	2.03E+01$^+$	2.13E+01$^+$	2.12E+01$^+$	2.12E+01$^+$	2.13E+01$^+$
f_{15}	1.88E+02	**1.16E+02**	2.86E+02	1.58E+02	1.76E+02	1.92E+02	2.28E+08$^+$	2.11E+02	1.24E+03$^+$	1.38E+02	6.45E+07$^+$	3.92E+02	1.61E+04$^+$	1.92E+02	1.74E+02	1.13E+03$^+$
f_{16}	9.86E+01$^+$	9.24E+01	9.68E+01	9.37E+01	9.32E+01	9.54E+01	2.88E+03$^+$	9.87E+01$^+$	9.70E+01	**9.10E+01**	1.48E+02$^+$	9.49E+01	3.28E+02$^+$	9.72E+01	9.42E+01	9.89E+01$^+$
f_{17}	2.71E-19	4.34E-19	3.09E-09$^+$	2.71E-19	**1.08E-19**	3.25E-19	1.22E+01$^+$	1.91E-01$^+$	1.80E-11$^+$	1.36E-19	8.95E+00$^+$	3.25E-19	1.07E+00$^+$	7.82E-02$^+$	2.71E-19	8.38E-02$^+$
f_{18}	2.20E-15	5.96E-19	1.60E-06	3.39E-13	4.58E-14	1.82E-14	2.90E+01$^+$	1.40E-12$^+$	2.55E-10$^+$	**2.17E-19**	1.05E-06$^+$	4.34E-19	1.42E+00$^+$	3.09E-06$^+$	1.76E-13	9.96E-01$^+$
f_{19}	**2.19E+01**	3.81E+02$^+$	6.96E+01	3.09E+02$^+$	3.92E+02$^+$	2.40E+02	6.91E+02$^+$	1.75E+02	2.57E+03$^+$	4.86E+02$^+$	5.51E+02$^+$	1.62E+03$^+$	7.17E+02$^+$	3.65E+02$^+$	3.91E+02$^+$	1.32E+03$^+$
f_{20}	7.78E+02$^+$	6.74E+02$^+$	5.60E+02	4.40E+02	6.63E+02	5.61E+02	8.26E+02$^+$	6.98E+02	1.02E+03$^+$	8.60E+02$^+$	7.84E+02$^+$	8.05E+02$^+$	6.30E+02	**3.76E+02**	4.49E+02	7.95E+02$^+$
f_{21}	5.59E+04$^+$	5.27E+04	**4.75E+04**	5.32E+04	6.44E+04$^+$	5.23E+04	5.83E+04$^+$	4.88E+04	6.02E+04$^+$	6.58E+04$^+$	5.53E+04$^+$	5.94E+04$^+$	5.52E+04$^+$	5.75E+04$^+$	5.58E+04$^+$	5.88E+04$^+$
f_{22}	5.95E+04$^+$	5.84E+04	**5.69E+04**	5.76E+04	7.30E+04$^+$	5.72E+04	6.79E+04$^+$	7.36E+04$^+$	6.09E+04$^+$	7.36E+04$^+$	6.77E+04$^+$	5.97E+04$^+$	7.25E+04$^+$	5.90E+04$^+$	6.27E+04$^+$	5.92E+04$^+$
f_{23}	9.36E+01	9.14E+01	7.35E+01	9.62E+01	6.79E+01	9.28E+01	1.25E+02$^+$	**4.59E+01**	1.44E+02$^+$	7.53E+01	1.34E+02$^+$	1.21E+02$^+$	8.60E+01	5.90E+01	1.00E+02	1.11E+02$^+$
f_{24}	4.16E-01	1.26E+00	6.45E-01	1.52E+00	4.14E+00$^+$	1.32E+00	2.61E+00$^+$	4.16E+00$^+$	2.13E+00$^+$	4.17E+00$^+$	4.18E+00$^+$	1.65E+00	4.18E+00$^+$	**8.22E-03**	1.43E+00	7.96E-01
f_{25}	5.76E-01$^+$	7.03E-01$^+$	4.62E-01	5.23E-01$^+$	5.49E-01$^+$	4.46E-01	6.91E-01$^+$	6.04E-01$^+$	6.02E-01$^+$	5.41E-01$^+$	6.02E-01$^+$	6.06E-01$^+$	5.89E-01$^+$	**4.05E-01**	4.66E-01	6.19E-01$^+$
f_{26}	3.63E-01$^+$	8.34E-01$^+$	3.48E-01$^+$	3.28E-01	3.79E-01$^+$	**3.15E-01**	1.00E+00$^+$	3.76E-01$^+$	3.50E-01	3.74E-01$^+$	4.03E-01$^+$	3.47E-01	4.04E-01$^+$	3.43E-01	3.30E-01	3.93E-01$^+$
f_{27}	**3.30E+01**	2.44E+02$^+$	1.00E+02	1.13E+02	7.90E+01	1.36E+02	4.34E+02$^+$	2.21E+02$^+$	2.57E+02$^+$	7.81E+01	2.12E+03$^+$	2.25E+02$^+$	2.57E+02$^+$	1.90E+02$^+$	9.88E+01	2.43E+02$^+$
f_{28}	1.87E+02	3.28E+02$^+$	2.07E+02	2.69E+02	**1.84E+02**	2.33E+02	3.42E+07$^+$	4.26E+02$^+$	3.54E+02$^+$	2.23E+02	6.41E+07$^+$	2.98E+02$^+$	1.46E+03$^+$	2.83E+02$^+$	3.36E+02$^+$	5.88E+02$^+$
f_{29}	**4.34E+01**	2.08E+02$^+$	6.97E+01	1.69E+02	1.94E+02$^+$	1.50E+02	4.33E+02$^+$	1.48E+02	7.56E+02$^+$	3.01E+02$^+$	4.62E+02$^+$	6.22E+02$^+$	3.01E+02$^+$	3.00E+02$^+$	2.75E+02$^+$	8.75E+02$^+$
f_{30}	1.20E-10	2.07E-06$^+$	2.18E-03$^+$	5.44E-24	**0.00E+00**	7.64E-07	5.63E+01$^+$	5.29E-06	1.29E+03$^+$	2.06E-01	**0.00E+00**	6.12E+01$^+$	5.18E+00$^+$	4.70E-01$^+$	4.44E-16	5.21E+01$^+$
f_{31}	1.93E+02	5.04E+02$^+$	2.32E+02	3.00E+02	1.11E+02	2.99E+02	5.26E+02$^+$	3.97E+02$^+$	5.15E+02$^+$	**1.10E+02**	2.32E+02	4.63E+02$^+$	4.57E+02$^+$	3.45E+02	3.15E+02	5.06E+02$^+$
f_{32}	5.06E+02	7.51E+02$^+$	4.76E+02	6.01E+02	2.38E+02	5.89E+02	9.83E+02$^+$	6.19E+02	7.72E+02$^+$	**1.75E+02**	1.77E+02	7.19E+02$^+$	8.92E+02$^+$	5.67E+02	7.58E+02$^+$	9.72E+02$^+$
f_{33}	**8.91E+01**	1.65E+02$^+$	9.24E+01	1.41E+02	1.38E+02	1.41E+02	2.34E+02$^+$	1.02E+02	3.19E+02$^+$	1.49E+02	1.76E+02$^+$	2.42E+02$^+$	1.77E+02$^+$	1.57E+02$^+$	1.72E+02$^+$	3.18E+02$^+$
f_{34}	9.78E-13	1.05E+00$^+$	1.43E-03$^+$	2.30E-18	1.05E+00$^+$	3.94E-07	9.57E+00$^+$	2.57E+00$^+$	7.53E+01$^+$	3.15E+00$^+$	**0.00E+00**	6.48E+01$^+$	1.48E+01$^+$	9.16E+00$^+$	1.05E+00$^+$	7.59E+01$^+$
f_{35}	2.59E+04$^+$	**2.49E+04**	2.51E+04	2.52E+04$^+$	2.51E+04	**2.49E+04**	3.40E+04$^+$	2.51E+04	2.82E+04$^+$	2.53E+04$^+$	2.52E+04$^+$	2.56E+04$^+$	2.54E+04$^+$	2.51E+04	2.52E+04$^+$	2.70E+04$^+$
f_{36}	3.13E+05	5.68E+05$^+$	**2.53E+05**	2.98E+05	5.67E+06$^+$	2.79E+05	6.93E+05$^+$	3.79E+05$^+$	4.51E+05$^+$	1.64E+08$^+$	2.50E+06$^+$	3.88E+05$^+$	7.08E+05$^+$	2.86E+05	3.60E+05$^+$	6.83E+05$^+$
f_{37}	1.32E+02	1.13E+02	**9.85E+01**	1.26E+02	1.32E+02$^+$	1.20E+02	1.29E+02$^+$	1.32E+02$^+$	1.20E+02	1.32E+02$^+$	1.30E+02$^+$	1.02E+02	1.33E+02$^+$	1.24E+02	1.20E+02	1.33E+02$^+$

* In the row "Wilcoxon test", we use the symbol $\approx$ to indicate those cases in which the Wilcoxon test with confidence at 0.95 using Bonferroni's correction did not return a p-value lower than $\alpha = 0.05$; and the symbol $+$ to indicate those cases in which the difference was significant (i.e., $p < \alpha$).

Table 8.8 Continued.)

f#	PSO-X_{all}	PSO-X_{uni}	PSO-X_{mul}	PSO-X_{hyb}	PSO-X_{soco}	PSO-X_{cec}	GauPSO$_{tnd}$	ResPSO$_{tnd}$	ERiPSO$_{dft}$	FinPSO$_{tnd}$	FraPSO$_{dft}$	LcRPSO$_{dft}$	HiePSO$_{tnd}$	IncPSO$_{tnd}$	StaPSO$_{tnd}$	SPSO11$_{tnd}$
f_{38}	3.28E+05	5.24E+05	3.47E+05	5.33E+05	7.48E+05$^+$	5.68E+05	9.86E+05$^+$	8.18E+05$^+$	5.32E+05	8.83E+05$^+$	**3.02E+05**	7.01E+05$^+$	8.51E+05$^+$	3.98E+05	6.69E+05$^+$	7.50E+05$^+$
f_{39}	9.43E+00$^+$	1.76E+01$^+$	7.03E+00	7.20E+00	8.61E+00$^+$	**6.96E+00**	2.13E+04$^+$	9.06E+00$^+$	1.56E+01$^+$	8.37E+00$^+$	1.10E+01$^+$	1.15E+01$^+$	6.60E+01$^+$	7.10E+00	7.13E+00	9.59E+00$^+$
f_{40}	1.74E+01	3.22E+01$^+$	1.87E+01	2.14E+01	**1.33E+01**	2.31E+01	5.40E+02$^+$	2.04E+01	1.07E+02$^+$	1.62E+01	2.47E+01	4.30E+01$^+$	3.87E+01$^+$	2.60E+01$^+$	2.10E+01	1.01E+02$^+$
f_{41}	3.46E+02$^+$	4.00E+02$^+$	3.37E+02	2.98E+02	3.51E+02$^+$	**2.24E+02**	3.04E+02	4.20E+02$^+$	9.48E+02$^+$	3.36E+02$^+$	7.45E+02$^+$	8.54E+02$^+$	4.06E+02$^+$	4.08E+02$^+$	4.00E+02$^+$	6.14E+02$^+$
f_{42}	2.00E+02$^+$	2.23E+02$^+$	1.20E+02$^+$	1.09E+02	2.77E+02$^+$	**9.44E+01**	2.76E+02$^+$	3.59E+02$^+$	4.61E+02$^+$	2.95E+02$^+$	5.85E+02$^+$	4.00E+02$^+$	3.27E+02$^+$	1.16E+02$^+$	2.40E+02$^+$	3.37E+02$^+$
f_{43}	2.25E+02	3.16E+02	3.01E+02$^+$	2.38E+02	3.09E+02$^+$	2.35E+02	3.61E+02$^+$	3.32E+02$^+$	4.86E+02$^+$	3.23E+02$^+$	5.07E+02$^+$	4.34E+02$^+$	3.55E+02$^+$	**1.65E+02**	2.54E+02	3.67E+02$^+$
f_{44}	9.55E+02$^+$	9.34E+02	**9.25E+02**	**9.25E+02**	9.29E+02	9.27E+02	9.51E+02$^+$	9.44E+02$^+$	1.12E+03$^+$	9.38E+02$^+$	1.22E+03$^+$	1.08E+03$^+$	9.37E+02$^+$	**9.25E+02**	9.40E+02$^+$	9.38E+02$^+$
f_{45}	9.56E+02$^+$	9.34E+02$^+$	9.25E+02	9.28E+02	9.29E+02	9.25E+02	9.50E+02$^+$	9.39E+02$^+$	1.13E+03$^+$	9.37E+02$^+$	1.21E+03$^+$	1.10E+03$^+$	9.39E+02$^+$	**9.17E+02**	9.39E+02$^+$	9.36E+02$^+$
f_{46}	9.59E+02$^+$	9.33E+02$^+$	9.25E+02	9.28E+02	9.29E+02	9.25E+02	9.50E+02$^+$	9.39E+02$^+$	1.12E+03$^+$	9.37E+02$^+$	1.21E+03$^+$	1.10E+03$^+$	9.39E+02$^+$	**9.17E+02**	9.40E+02$^+$	9.36E+02$^+$
f_{47}	8.02E+02	1.02E+03	1.01E+03	1.02E+03	1.01E+03	1.02E+03	1.02E+03$^+$	1.01E+03	1.28E+03$^+$	6.57E+02	1.31E+03$^+$	1.23E+03	**6.34E+02**	1.01E+03	9.42E+02	1.02E+03
f_{48}	9.68E+02$^+$	9.58E+02	9.34E+02	**9.09E+02**	9.22E+02	9.18E+02	**9.09E+02**	9.80E+02$^+$	1.17E+03$^+$	9.70E+02$^+$	1.37E+03$^+$	1.11E+03$^+$	9.57E+02$^+$	9.56E+02	9.20E+02	9.92E+02$^+$
f_{49}	9.76E+02	1.02E+03$^+$	1.02E+03	1.02E+03	1.01E+03	1.02E+03	1.02E+03	1.01E+03	1.28E+03$^+$	9.50E+02	1.04E+03	1.26E+03$^+$	8.24E+02	1.02E+03	**7.29E+02**	1.02E+03
f_{50}	9.42E+02	9.84E+02	1.11E+03$^+$	9.57E+02	9.38E+02	9.85E+02	9.54E+02	9.39E+02	1.27E+03$^+$	**2.00E+02**	1.35E+03$^+$	1.30E+03$^+$	9.72E+02	1.15E+03	9.06E+02	9.79E+02
Wins	9	5.	6	6	7	**11**	1	1	1	8	5	2	1	8	1	0
Av.MED	4.02E+05	1.18E+05	3.63E+05	2.58E+05	9.43E+05	1.98E+05	7.16E+07	2.26E+05	1.35E+05	4.84E+06	3.51E+06	**8.32E+04**	6.22E+06	3.12E+05	5.45E+05	5.89E+05
Av.Ranking	6.38	7.6	5.66	5.7	6.44	**5.1**	13.46	9.1	12.02	7.66	11.44	10.5	11.4	7.16	7.18	12.4
Wilcoxon test	$\approx$	$\approx$	$\approx$	$\approx$	$\approx$		$+$	$+$	$+$	$\approx$	$+$	$+$	$+$	$\approx$	$\approx$	$+$

* In the row "Wilcoxon test", we use the symbol $\approx$ to indicate those cases in which the Wilcoxon test with confidence at 0.95 using Bonferroni's correction did not return a p-value lower than $\alpha = 0.05$; and the symbol $+$ to indicate those cases in which the difference was significant (i.e., $p < \alpha$).

Table 8.9: Average ranking (Av.Ranking) and average median error (Av.MEDerr) obtained by the six automatically generated PSO-X algorithms and ten other PSO algorithms in f_{1-12} (unimodal), f_{13-26} (multimodal), f_{27-50} (hybrid) and f_{rotated}.

f#		PSO-X_{all}	PSO-X_{uni}	PSO-X_{mul}	PSO-X_{hyb}	PSO-X_{soco}	PSO-X_{cec}	GauPSO$_{tnd}$	ResPSO$_{tnd}$	ERiPSO$_{dft}$	FinPSO$_{tnd}$	FraPSO$_{dft}$	LcRPSO$_{dft}$	HiePSO$_{tnd}$	IncPSO$_{tnd}$	StaPSO$_{tnd}$	SPSO11$_{tnd}$
f_{1-12}	Av.Ranking	5.92	6.92	7.75	6.75	5.5	**5.25**	15.33	9.08	9.17	7.75	9.67	7.92	11.83	8.83	8.58	12.8
	Av.MEDerr	1.51E+06	2.82E+05	1.34E+06	8.92E+05	3.28E+06	6.36E+05	2.76E+08	7.27E+05	3.63E+05	6.31E+06	3.53E+06	**1.39E+05**	2.57E+07	1.13E+06	2.07E+06	2.21E+06
f_{13-26}	Av.Ranking	5.88	7.12	**4.54**	5.23	7.08	4.85	13.9	8.88	11.92	8.35	10.77	10.04	12.35	6.77	6.85	12.65
	Av.MEDerr	1.05E+03	7.72E+02	**3.38E+02**	7.42E+02	2.53E+03	6.61E+02	1.52E+07	4.58E+03	3.01E+03	2.72E+03	4.30E+06	1.91E+03	4.09E+03	2.32E+03	1.37E+03	3.53E+03
f_{27-50}	Av.Ranking	**3.28**	6.16	3.76	3.6	5.36	5.5	9.36	8.34	11.24	8.68	11.24	12.18	11.92	10.3	11.1	14.28
	Av.MEDerr	1.27E+04	3.07E+04	**1.10E+04**	2.02E+04	2.44E+05	2.08E+04	1.42E+06	3.49E+04	2.67E+04	6.59E+06	2.66E+06	3.07E+04	4.94E+04	1.44E+04	2.82E+04	4.45E+04
f_{rotated}	Av.Ranking	7.45	6.91	5.64	5	7.36	**4.68**	12.09	10	10.86	8.77	14.18	9.59	11	5.64	6.82	10.9
	Av.MEDerr	8.19E+05	1.46E+05	7.28E+05	4.83E+05	1.79E+06	3.46E+05	1.51E+08	3.94E+05	1.97E+05	3.44E+06	1.93E+06	**7.53E+04**	1.40E+07	6.16E+05	1.12E+06	1.20E+06
f_{1-50}	Av.Ranking	6.2	7.54	5.54	5.5	6.34	**4.92**	13.28	8.94	11.98	7.66	11.44	10.48	11.34	6.9	7.1	12.3
	Av.MEDerr	3.69E+05	8.33E+04	3.27E+05	2.24E+05	9.09E+05	1.63E+05	7.16E+07	1.92E+05	1.01E+05	4.81E+06	3.47E+06	**4.91E+04**	6.19E+06	2.79E+05	5.12E+05	5.53E+05

* We show in boxes the rankings of the PSO variants that are better than, or as performing as, any of the automatically generated PSO-X algorithm.

Note that only IncPSO$_{tnd}$ and StaPSO$_{tnd}$ were capable of outperforming the results obtained by some of the PSO-X algorithms, specially in the rotated functions, where those two algorithms were as competitive as those automatically generated. However, in the case of the hybrid composition functions the results are quite compelling in favor of PSO-X, since even the worst automatically generated algorithm performed significantly better than any of the PSO variants. This is a very strong point in favor of our PSO-X algorithms not only because half of the functions in our benchmark set are hybrid compositions, but also because these kinds of functions are the hardest to solve and the most representative of real-world optimization problems.

According to Wilcoxon pair-wise tests between PSO-X_{cec} and the PSO variants using the data presented in Table 8.8, the median solution values obtained by FinPSO$_{tnd}$, StaPSO$_{tnd}$ and IncPSO$_{tnd}$ are not statistically different from PSO-X_{cec}. FinPSO$_{tnd}$ was the best performing of the PSO variants in the unimodal functions, and IncPSO$_{tnd}$ in the multimodal and rotated functions. The three PSO variants have some commonalities regarding their design, including that they all use low connected topologies (Von Nemann and ring) during most of their execution (see Table 8.4) and, in the case of FinPSO$_{tnd}$ and IncPSO$_{tnd}$, they both use the CCVUR. While our experimental results show that only IncPSO$_{tnd}$ and StaPSO$_{tnd}$ are clearly better than one of our algorithms (PSO-X_{uni}) across the whole benchmark set, the three PSO variants (FinPSO$_{tnd}$, StaPSO$_{tnd}$ and IncPSO$_{tnd}$) produced results that are competitive with the PSO-X algorithms in some specific classes of functions. Finally, it is worth pointing out that none of the default versions of the PSO variants that we included in our comparison (that is, ERiPSO$_{dft}$, FraPSO$_{dft}$ and LcRPSO$_{dft}$) performed as well as the ones that were configured with `irace` in terms of median solution quality. It is particularly interesting the case of StaPSO and FinPSO, whose performance improved dramatically after the configuration process. However, it is extremely common to see these two variants implemented with default parameters in many papers proposing and comparing new algorithms.

8.5.3 Are PSO-X implementations convergent?

Local convergence is one of the most salient characteristics of high-performing PSO implementations. It prevents unwanted roaming (which often results in particles leaving the search space) and allows particles to improve the initial solutions for any number of dimensions. Creating convergent implementation

using PSO-*X* is possible because (i) the value of the three main parameters of PSO (ω_1, φ_1 and φ_2) is limited within the theoretical region where local convergence is expected to occur,[9] and (ii) in PSO-*X* are implemented both a number of algorithm components that have been shown to result in particles local convergence (e.g., DNPP-spherical, $\text{Pert}_{\text{info}}$-Gaussian, $\text{Pert}_{\text{info}}$-Lévy, etc.) and strategies that limit the magnitude of the velocity vector (velocity clamping).

As it can be observed in Table 8.4 and 8.5, the six PSO-*X* algorithm that PSO-*X* automatically created use the two main algorithm components proposed for the LcRPSO (i.e., DNPP-rectangular and $\text{Pert}_{\text{info}}$-Gaussian).[10] In (Bonyadi and Michalewicz 2014), it was formally and experimentally demonstrated that LcRPSO is locally convergent because the $\text{Pert}_{\text{info}}$-Gaussian component satisfies the local convergence condition, which ensures that the mapping between the input vector $\vec{r}$ and the perturbed vector $\mathcal{N}(\vec{r}, \sigma_t)$ is located in any definable region of the search space—see (Bonyadi and Michalewicz 2014, Appendix 1) for the formal definition of this condition. Although our PSO-*X* algorithms are six different specializations of LcRPSO, by using a number of algorithm components that were found to be good design choices during the configuration process, they exhibit better performance than any of the variants considered in this study. We believe this shows the power of combining automatic configuration and component-based framework to create high-performing algorithm implementations.

8.6 Summary

In this chapter, we presented PSO-*X*, a flexible, automatically configurable framework that combines metaheuristic components and automatic configuration tools to create high performing PSO implementations. Six PSO algorithms were automatically created from the PSO-*X* framework and compared with ten well-known PSO variants published in the literature. The results obtained after solving a set of 50 benchmark functions with different characteristics and complexity showed that the automatically created PSO-*X* algorithms exhibited higher performance than their manually created counterparts.

In PSO-*X*, we have incorporated many relevant ideas proposed in the litera-

[9] In Section B.2 of Appendix B, we report the value of these parameters for the six PSO-*X* algorithms and the ten PSO variants and the conditions for order-1 stability.

[10] Note that, in addition to $\text{Pert}_{\text{info}}$-Gaussian, it is possible to use the $\text{Pert}_{\text{info}}$-Lévy component, since the Lévy distribution is a generalization of the Gaussian

ture for the PSO algorithm, including: different topologies, models of influence and ways of handling the population; several strategies to set the value of the algorithm parameters; a number of ways to construct and apply random matrices; and various kinds of distributions of particles positions in the search space. With PSO-*X*, we seek to provide a tool that can simplify the application of PSO to tackle continuous optimization problems, and also to bring clarity on the main design choices available when implementing it.

In addition to this, even though it was not among the goals for proposing PSO-*X*, it is indeed possible to use this framework to instantiate some of the "novel" metaphor-based metaheuristics that we have discussed in Chapter 6. For example, the *grey wolf optimizer* algorithm can be easily instantiated from PSO-*X* by using the DNPP-spherical component, the *firefly algorithm* by using the DNPP-standard, AC-extrapolated and MoI-fully informed components, the *bat algorithm* by using the AC-time-varying component, etc. In our opinion, the fact that these "novel" metaheuristics can be instantiated from PSO-*X* just by selecting specific metaheuristic components proposed for PSO shows, in a very compelling way, that the behaviors that inspired the "novel" metaheuristics were completely unnecessary from the beginning, as they did not bring anything new to the field. Unfortunately, the only function of the metaphors was to hide the fact that the proposed ideas had been already proposed in the PSO literature.

Finally, it is worth mentioning one obvious limitation in our work: since PSO is a intensively studied algorithm with hundreds of variants, including in PSO-*X* the totality of the ideas proposed for this algorithm is challenging. Hence, a continuous effort must be done to keep adding new algorithms to PSO-*X* so that implementations remain competitive with the state of the art.

Part IV

Discussion

Chapter 9

Too Many Metaphors, Too Little Automatic Design: Is the Field of Metaheuristics Moving in the Right Direction?

In this chapter, we address various aspects of the field of metaheuristics in an attempt to understand whether the field is moving in the right direction or not. We present arguments for both a positive and a negative answer. We start by discussing the difficulties of finding metaphors that lead to truly innovative algorithms. Then, we discuss the rationale often presented by the authors of "novel" metaphor-based algorithms as their motivation to propose more algorithms of this kind, which is based on a wrong understanding of the No-Free-Lunch theorems for optimization. We conclude this chapter by discussing how to improve three foundational aspects of the field, namely the focus of the research, the way metaheuristics implementations are benchmarked, and the creation of new metaheuristics.

9.1 Why Some Metaphors Work While Others Do Not?

In the period of time comprised between the late 1960s and the beginning of 2000s, the field of metaheuristics was characterized by a rapid growth in the number of new methods proposed to tackle complex optimization problems

and by many experimental and theoretical studies of such methods investigating their properties and ways to extend their capabilities (Corne et al. 1999; Gendreau and Potvin 2019; Sörensen et al. 2018). The field also witnessed the introduction of a few very successful techniques inspired by natural phenomena such as *evolutionary algorithms* (Fogel et al. 1966; Holland 1975; Rechenberg 1971; Schwefel 1977, 1981), where the inspiration is the phenomenon of evolution by natural selection and survival of the fittest; *simulated annealing* (Černý 1985; Kirkpatrick et al. 1983), inspired by the metal annealing process whereby atoms reorganize themselves so as to minimize an energy function; *ant colony optimization* (Dorigo 1992b; Dorigo et al. 1991b), inspired by the foraging behavior of ants; and *particle swarm optimization* (Kennedy and Eberhart 1995), inspired by the dynamics and social interaction in bird flocks. These methods not only attracted the attention of scientists and practitioners interested in solving relevant problems for which other methods fail to provide satisfactory results, but also led to one of the most widespread beliefs in the field, that is, that "nature is a never ending source of inspiration" to tackle complex optimization problems.

Even though there are some examples of natural behaviors that have been useful to design new and efficient optimization algorithms, the truth is that finding a natural behavior that leads to both *useful* and *novel* ideas on how to solve optimization problems turns out to be very difficult. The first difficulty comes from the fact that one can be blinded by the desire of introducing a successful novel algorithm and not realize that the considered behavior is irrelevant for the purpose of designing of an optimization algorithm. A clear example of this is given by the "novel" algorithms analyzed in Chapters 5 and 6. Most of these "novel" algorithms are based on interesting behaviors exhibited by different species of animals; however, when these behaviors were abstracted as concepts to be used in an optimization algorithms, they turned out to be either totally *useless* in optimization or to lead to the same ideas that were already proposed in the past.

Indeed, the second difficulty is the difficulty of finding something that is truly novel in the field of optimization. Clearly, there is no guarantee whatsoever that just by looking for inspiration in nature, or elsewhere, one can find something that, when abstracted as an optimization algorithm, is both useful and *novel*. In fact, it can be the case that an original, scientifically motivated, and properly abstracted behavior results in an optimization algorithm whose design is equivalent to one already proposed in the literature. An example of this is the *biogeography-based optimization* (Simon 2008), which, as its author showed in

a later study (Simon et al. 2011), is "a generalization of a genetic algorithm with global uniform recombination", an algorithms first proposed in the 1960s.

9.2 Metaphor-based Algorithms Motivated by Theoretical Research

In papers proposing "novel" metaphor-based metaheuristics, it is common to find a mention of the No-Free-Lunch (NFL) theorems formulated by Wolpert and Macready (1997) as the theoretical foundation motivating the metaheuristic, such as in the following examples:

> *"Obviously, the No Free Lunch theorem makes this field of study highly active which results in enhancing current approaches and proposing new meta-heuristics every year. This also motivates our attempts to develop a new meta-heuristic with inspiration from grey wolves."*
> (Mirjalili et al. 2014, pp. 46–47)

> *"Some of the most popular algorithms in this field are: Genetic Algorithms (GA), Particle Swarm Optimization (PSO), Ant Colony Optimization (ACO), Differential Evolution (DE), Evolutionary Programming (EP). Although these algorithms are able to solve many real and challenging problems, the so-called No Free Lunch theorem allows researchers to propose new algorithms."*
> (Mirjalili 2015b, p. 81)

> *"According to the "no-free-lunch" (NFL) theory, it is difficult to employ a single meta-heuristic algorithm in striving to solve all possible optimization problems."* ... *"This has been a motive for the researchers in this field, as well as ourselves, to look for new and innovative nature-inspired methods to solve and show superior scores on the current and new hard real-life problems. The door is still open, and here we present a novel meta-heuristic algorithm based on human behavior with the very famous tale of Ali Baba and the forty thieves, as our inspiration targeting numerical optimization problems."*
> (Braik et al. 2022)

However, even though the main implication of the NFL is that "all search heuristics have the same performance when averaged over the uniform distribution over all possible objective functions", the assumptions that lead to this

implication have been criticized as *restrictive* and *unrealistic*. Indeed, it has been demonstrated that the NFL is unrealistic in black-box optimization scenarios (i.e., where the objective function can be queried but its definition remains unknown) and that the theorems do not hold for multi-objective problems and continuous domains—being this latter the domain of application of the three algorithms in the quotations above. A detailed discussion of the implications of the NFL in problems that are relevant in practice can be found in (Auger and Teytaud 2007, 2010) and in the references cited in these papers.

However, beyond the real implications of the NFL, it is obvious that faster and more performing optimization algorithms are necessary to tackle the ever more challenging optimization problems arising in all fields and disciplines. For the vast majority of the researchers in the field of metaheuristics, developing better optimization algorithms requires to understand the weaknesses and strengths of the existing ones before finding new sources of inspiration. Unfortunately, rather than contributing to improve the state of the art, the hundreds of "novel" metaphor-based algorithms already published in the literature are leading the field astray, creating confusion, and making the use of unscientific practices to seem *normal* in a scientific field. Moreover, as illustrated in the quotations above, the authors of metaphor-based metaheuristics seem to think that advancing the field means introducing more and more absurd metaphors and develop more and more "novel" algorithms, despite the growing evidence clearly pointing to the fact that this approach is greatly inefficient and has many negative consequences.

9.3 Where Do We Go From Here?

In their 2017 contribution to the Handbook of Metaheuristics, "A history of metaheuristics", Kenneth Sörensen, Marc Sevaux, and Fred Glover (2018) posed as a possibility that the next transition in the development of metaheuristics would be towards a *scientific period*. Even though it seems paradoxical to call for a scientific period in a scientific field, the reason for this is both unfortunate and simple: there is an increasing amount of research published in the literature that is anything but scientific. Most of this unscientific research is related to the trend of the "novel" metaphor-based metaheuristics that we discussed in Chapter 4. This trend has been particularly difficult to stop due to the fact that there is a relatively large active community "researching" these kinds of algorithms.

Sadly, and to make matters worse, a pretty high number of papers have been published that present the "novel" metaheuristics in a positive way, ignoring the well-founded criticisms and/or taking such criticisms out of context.[1]

We strongly believe that, once the metaheuristics community as a whole begins to look at the trend of the "novel" metaphor-based metaheuristics and related research with scientific eyes, the trend will soon be stopped, many published papers proposing "novel" metaheuristics will be withdrawn from journals and conferences, and the whole thing will become a cautionary tale of something that we should avoid in the future. Another reason to call for a more scientific view is to try and unify an increasing body of research that seems to be moving in opposite directions. Adopting a scientific view as baseline for a field that is still expanding due to an active research community is the *best* way to prevent the (re)appearance of detrimental trends in which personal beliefs can override rational thinking. In addition to the aforementioned paper by Sörensen et al. (2018), there have been a few other calls for the adoption of a more scientific view in the field.

One of the most recent attempts to steer the metaheuristics community towards a more scientific direction has been the "*Metaheuristics in the Large*" community project (Swan et al. 2022). In this project, several prominent researchers presented their long-term vision for the field, which consists of three main conceptual underpinnings: (i) extensible and re-usable framework templates—which refers to the concept of modern MSFs as described in Chapter 7 of this thesis; (ii) white-box problem descriptions—which refers to the use of analytic information to guide the metaheuristic selection/construction in an informed manner; and (iii) remotely accessible frameworks, components and problems—which refers to the creation of service-oriented architectures that enable the widespread re-use of data and programs.

Another attempt to refocus the course of the field is a recent open letter titled "*Metaphor-based metaheuristics, a call for action: the elephant in the room*" (Aranha et al. 2022), which brought together around 100 researchers in the field that subscribed to the goal of putting a limit to the publication of "novel" metaphor-based metaheuristics by adopting concrete actions, such as calling for scientific journals to establish clear editorial policies concerning how to manage articles presenting this type of metaheuristics. This open letter has been one of

[1] See, e.g., K. Rajwar, K. Deep and S. Das. An exhaustive review of the metaheuristic algorithms for search and optimization: taxonomy, applications, and open challenges. *Artificial Intelligence Review*. Springer, 2023.

the most compelling efforts conducted so far to increase the awareness about the problem of the "novel" metaphor-based metaheuristics.

While the vision presented in these documents has been an important step towards steering the field into a healthier direction, there is still much to be done in this regard. In particular, it seems that more efforts are needed to bring together the metaheuristics community to address the various issues that have remained unresolved for years. The next sections discuss three fundamental ways that we suggest might prove to be decisive in removing from the field of metaheuristics the less scientific approaches. In particular, we discuss the need (i) to increase the amount of research that is either experimentally- or theoretically-driven; (ii) to improve the way metaheuristics are bechmarked; and (iii) to change the current mainstream approach to create metaheuristics.

9.3.1 Rethinking the Focus of the Research

As a field of study, metaheuristics are first and foremost an applied science that deals with the design and application of optimization algorithms[2] that work well (often much better than well) regardless of the complexity of the problems. As such, the field has always had a strong bias towards application-oriented research and has extensively used "competitive testing"[3] to make claims about the algorithms' performance (Hooker 1996). However, even though competitive testing allows to answer the question *"Which of the considered algorithms will provide the best solution quality for the problem at hand?"*, it does not allow to delve deeper into the reasons that lead to the observed results. Therefore, when it comes to advancing the field and to increasing our knowledge of *why* some techniques work well on some problems and not on others, it is key to reduce the asymmetry between the amount of research that is application-oriented and the one that is experimentally- or theory-driven.

Studying the interplay between the computational models underlying a metaheuristic and its performance on different problems classes is important not only from a research perspective, but also from a practical one. Exper-

[2]In the remainder of this chapter, when discussing general aspects of the field, we use the more general term "optimization algorithms" instead "metaheuristics".

[3]Competitive testing consists in evaluating the performance of the compared algorithms so as to determine the "winner" of the competition, that is, the algorithm that performs better than any of its competitors on a given set of problem instances and for a given performance measure (Bartz-Beielstein et al. 2020). However, it does not provide any information about what causes the differences in performances.

imental and theoretical analyses allow, for example, (i) to understand how extensible are the ideas involved in a metaheuristic, and therefore, to know how they can be used to address other problems; (ii) to guide the development of new metaheuristic components that can further improve the metaheuristic performance or solve issues that have been identified so far; and (iii) to define guidelines for creating metaheuristic implementations, i.e., useful indications of the different ways in which metaheuristic components can be combined, so that new designs can be created and tested more easily.

Fortunately, in the last few years, things seem to have been changing for the better in this area, as it is nowadays much more frequent to find application-oriented papers that introduce new optimization algorithms motivated by empirical evidence or theoretical findings. Unfortunately, there are still many important challenges to be overcome that concern the way research itself is conducted in this field, starting by the methodological practices that we use to draw conclusions from experimental studies.

9.3.2 Rethinking the Way We Benchmark Metaheuristics

It was Carl Sagan who popularized the phrase "extraordinary claims require extraordinary evidence" (Sagan 1979). In the field of metaheuristics, there is great variety in the way evidence (i.e., data) is collected and used to draw conclusions about the metaheuristics' performance—a process commonly referred to as *benchmarking* (Bartz-Beielstein et al. 2020). For example, at one end of the spectrum there are articles about the "novel" metaphor-based algorithms, that, although do not lack extraordinary claims, are often textbook examples of poor scientific practice (see the discussion in Chapter 4). At the other end of the spectrum there are researchers creating new tools to evaluate the metaheuristics' performance (Eftimov et al. 2020) and investigating new methodologies to compare metaheuristics based on modern techniques, such as the use of deep statistical analyses (Eftimov et al. 2017).

In general, even though there are a number of state-of-the-art methodologies and tools available in the literature to evaluate optimization algorithms (Derrac et al. 2011; Eftimov et al. 2017; R Development Core Team 2008; Sheskin 2011), they are not always used to evaluate the metaheuristics' performance. For example, it is still frequent to see structurally biased metaheuristics that are evaluated on biased tests sets (Kudela 2022) and flawed experimental methodologies that present unfair comparisons, do not guarantee reproducibility, and neglect

important performance metrics (Bartz-Beielstein et al. 2010; Campelo and Takahashi 2019; García-Martínez et al. 2017; Hooker 1994, 1996; López-Ibáñez et al. 2021). These issues, however, happen more often on some publication venues than in others. Therefore, to really tackle these issues, higher scientific standards would need to be implemented and systematically enforced by all the different outlets in which the literature of metaheuristics is published.

Recently, a number of researchers came together and put forward a set of guidelines and best practices to benchmark optimization algorithms with the aim of addressing the poor benchmarking practices often used in the field (Bartz-Beielstein et al. 2020). Among the guidelines and best practices they have proposed are: (i) clearly specifying the goal of the benchmark study and design it accordingly; (ii) using benchmarks that are comprehensive on both the size, difficulty and diversity of the problems; (iii) using manual or automatic techniques to configure the parameters of the metaheuristics; (iv) using sound statistical methodologies in order to decide what experiments should be conducted, how many times each experiment should be repeated, which data should be gathered and how it should be processed, analyzed, interpreted, and presented; and (v) avoiding generalizing the results without having enough evidence to do it or without defining clear bounds within which such generalization apply.

It is impossible to overstate the importance that the research focused on improving the way metaheuristics are compared and evaluated has in this field, which involves the often underestimated effort of creating/updating testbeds on a regular basis in order to reflect the complexity found in ever changing realistic scenarios (Hansen et al. 2021). However, moving from theory to practice has proven to be quite challenging for a field that is experimental in nature and has been focused mostly on testing metaheuristics as if they were *horses in a race* for most of its history. Indeed, despite the many efforts devoted to improve the experimental practices used in the field (Bartz-Beielstein et al. 2010; Campelo and Takahashi 2019; García-Martínez et al. 2017; Hooker 1994, 1996), adopting such practices has not yet being widely done, particularly in some of the venues where "novel" metaphor-based metaheuristics are regularly published.

9.3.3 Rethinking the Way We Create Metaheuristics

Based on what we have discussed in this thesis, it seems only natural to argue in favor of moving from manual to automatic design as the main way to

create metaheuristics. This includes promoting the research that focuses on the creation of flexible, automatically configurable MSFs that can be extended with new metaheuristic components so that other researchers/practitioners can use them in different contexts. Additionally, it is key to promote the research on automatic design methods and their application to develop implementations that are tailored for the specific needs of the user. The long-term goal of this approach is to fully automatize the process of creating metaheuristics, such that, when a new problem arises, an effective metaheuristic implementation can be automatically created in a timely and unbiased way.

In addition to adopting the automatic design paradigm as the main way to create new metaheuristics, we also advocate here for harnessing the advances in machine learning (ML) to further boost the development of performing metaheuristics. Several research trends have emerged that explore ways of integrating ML into metaheuristics (Gambella et al. 2021; Karimi-Mamaghan et al. 2022; Song et al. 2019; Talbi 2021). Different levels of integration have been identified in the literature (Karimi-Mamaghan et al. 2022; Talbi 2021), such as *problem level integration*—where ML can aid in recomposing the objective function and constraints of the problem or in decomposing the search space; *high-level* (algorithm-level) *integration*—where ML is used to select a suitable algorithm from an algorithms portfolio; and *low-level* (component-level) *integration*—where ML is used to automate the task of selecting and fine-tuning the algorithm components that perform best for a particular problem.

In recent years, metaheuristics designers have also begun to make use of data analytics tools (e.g., functional ANOVA (Hooker 2012), forward selection (Hutter et al. 2013) and ablation (Fawcett and Hoos 2016)), which are at the core of data science, to obtain knowledge about the algorithms' performance from the data collected during the design process. The productive synergy created between ML and metaheuristics has resulted not only in new ways to design and implement increasingly effective algorithms, but also in new ways to study these techniques and understand why specific designs are performing whereas others are not.

9.4 Summary

In this chapter, we tried to answer the question of whether the field of metaheuristics is moving in the right directions or not. To us, as well as to many other

researchers in the metaheuristics community, the right direction is one where there is a balance among the different type of research (application-oriented and experimentally- and theoretically-driven), where new ideas are properly tested and compared, where the more efficient and less biased approaches replace those that are inefficient and more biased, and where the use of unscientific practices is either nonexistent or rare. Unfortunately, the issue of the "novel" metaphor-based metaheuristics is not only pushing the field of metaheuristics away from these ideals, but also creating many new problems in its wake that the metaheuristics community will have to sort out. Regardless, there are many reasons to be optimistic about the future of this field and to believe that the more robust and scientific approaches will win out over the less scientific ones.

Chapter 10

Conclusions and Future Work

10.1 Conclusions

In the last decades, metaheuristics have been the method of choice to find approximate solutions to many difficult optimization problems. However, even though they have allowed important advances in the optimization field, the way the majority of metaheuristics are created to this day is still the same as in the early days—that is, they are the result of a time-consuming, error-prone process, in which a human designer manually creates the different components to be used in the metaheuristic implementation. Although this way of creating metaheuristics has been successful in the past and, in a few cases, involves the use of metaphors from naturally occurring optimization behaviors, it is now time to move towards a new way of creating metaheuristics that avoids the pitfalls of manual design. In this thesis, we have examined in detail an alternative approach called automatic design, and we have presented strong arguments to move away completely from the approach of finding inspiration in other fields of knowledge to manually develop new metaheuristic implementations.

After the introductory Chapter 1, in which we described the motivation for this research work, outlined the structure of the thesis and gave a preview of the research contributions, we began this thesis in Chapter 2 by giving a short introduction to optimization and metaheuristics. First, we described the set of optimization problems that are difficult to solve—i.e., those that cannot be efficiently approached with exact or analytical methods—as these are the type of problems for which heuristics and metaheuristics are commonly applied. Then, we presented a number of successful metaheuristics inspired by natural and social behaviors, which are also among the best performing ones available

in the literature. In doing so, we intended to show that, historically, there has been a *reason* for using metaphors to devise truly innovative metaheuristics.

Chapter 3 was dedicated to present and contrast the two main approaches used to create new metaheuristics: manual design and automatic design. The automatic design of metaheuristics is based on the use of a component-based view to define a metaheuristic design space and on the application of automatic configuration tools to explore many different designs until one is found that satisfies the needs of the user. The research done on the automatic design of metaheuristics has already led to a number of metaheuristic software frameworks that enable the use of automatic configuration tools to design high-performing implementations in a very efficient way, such as PSO-*X*, whose advantages were shown experimentally later in the thesis.

In Chapter 4, we elaborated on the trend of the "novel" metaphor-based metaheuristics inspired by natural, artificial, and even supernatural behaviors, which since a few years is being strongly criticized due to the uselessness of the new metaphors to devise truly novel algorithms, and to the confusion they have created in the literature of the field. In this chapter, we also argued that this trend exists in part due to the prevalence of manual design as the main way to create metaheuristics.

In order to illustrate why these kinds of metaheuristics are so problematic, we presented a number of analyses of highly-cited "novel" metaheuristics that turned out to lack any novelty. In addition to study their mathematical models and compare them with those proposed in well-established metaheuristics, we analyzed the use of the metaphors and found that they are completely useless from the point of view of devising novel optimization algorithms. Chapter 5, which was dedicated to discrete optimization, presented the analysis of the *intelligent water drops* and five of its variants; Chapter 6, which was dedicated to continuous optimization, presented the analysis of the *grey wolf, moth-flame, whale, firefly, bat, antlion* and *cuckoo* algorithms. Unfortunately, these are just a few examples in a long list of hundreds of "novel" metaheuristics with similar problems.

In Chapters 7 and 8 of this thesis, we elaborated on the increasing application of the automatic design paradigm in the field of optimization and discussed why it is currently the most efficient way to create high-performing metaheuristics implementations, as well as a possible solution to the problem of the "novel" metaphor-based metaheuristics. In Chapter 7, we first discussed the different aspects surrounding the creation of flexible, automatically configurable meta-

heuristic software frameworks from which high-performing implementations can be instantiated. Then, we presented some of the most popular of these software frameworks and described how researchers and practitioners are using these tools to create metaheuristics implementations whose design has never been considered before in the literature. In Chapter 8, we presented PSO-*X*, the metaheuristic software framework for particle swarm optimization that we developed in the context of this research work. We experimentally showed that PSO-*X* can be used to instantiate high-performing PSO algorithms without the need to consider new metaphors.

We concluded this work in Chapter 9 trying to answer the question of whether the field is moving in the right direction based on the state of some foundational aspects of the field. First, we discussed some of the difficulties in finding truly useful behaviors that can be used as an inspiration for the design of novel metaheuristics, as well as the wrong practice of citing the No-Free-Lunch as theoretical basis for introducing a new metaphor. Then, we reflected on some foundational aspects of the metaheuristics research. Some of the conclusions from our reflection on these aspect are that, in order to continue advancing the field of metaheuristics, we need to:

- continue increasing the amount of research that is experimentally- or theoretically-driven rather than focusing only on purely application-driven research;
- use state-of-the-art benchmarking practices to evaluate and compare metaheuristics and not only use the so-called "competitive testing"; and
- use modern tools to automatically create high-performing metaheuristic implementations.

This doctoral thesis was focused on the last of the three foundational aspects, which argues for fundamentally changing the way metaheuristic implementations are created. Fortunately, the area of automatic design is growing rapidly and it seems only a matter of time before modern, automatic design methods become widespread and replace manual design as the mainstream approach to design new metaheuristics. This, we believe, will help to put a definitive end to the trend of the "novel" metaphor-based metaheuristics and will have a positive, long-lasting effect on the way we see, understand and apply these optimization algorithms.

10.2 Future Work

Although there is already a vast diversity of metaheuristic techniques and mechanisms to create new designs, there are still many opportunities in the field. In particular, we consider promising to devote efforts (i) to create modeling frameworks that allow to better characterize metaheuristics, which have already shown great potential to quantify the similarity between metaheuristics (Armas et al. 2022); (ii) to develop advanced ways to benchmark metaheuristics, such as the creation of readily accessible statistical tools (Eftimov et al. 2017, 2020); and (iii) to extend the existing ecosystem of metaheuristic software frameworks and investigate ways to increase the re-usability of their components (Swan et al. 2019).

Modeling frameworks and metaheuristics characterization. One of the main issues in the metaheuristics research field is the lack of a comprehensive framework that allows to readily characterize metaheuristics and the relationships that exist among them and with other optimization techniques. There are different approaches that can be considered to address this issue, such as the use of ontologies to represent the concepts that belong to different metaheuristic classes, and the use of large language models—e.g., GPT, Turing NLG, and BERT—to extract information from the ontologies. Using ontologies and large language models it should be possible not only to describe a metaheuristic in great detail and see how it is related to other techniques using different levels of abstraction, but also to have an efficient way to obtain a quick classification of a metaheuristic implementation and its degree of novelty.

Advance metaheuristics benchmarking. Another important research direction for the future is to find ways to improve the way we benchmark metaheuristics. An interesting approach in this direction is the use of the so-called meta-learning and continuous learning approaches, where a machine learning algorithm is used to extract features both from the benchmark problems and from the metaheuristic implementations in order to link the type of problems for which a particular metaheuristic design can obtain good results. This approach can be used to create a large database of metaheuristics designs and performance assessments that can be (re)used in future benchmarking studies. However, in order for this approach to be truly successful, the metaheuristics community would have to adopt sound data management practices that guarantee the trustworthiness of the collected data.

Integration of metaheuristic software frameworks. In the future, we can expect to see a much higher degree of integration of the existing different optimization tools and of the mechanisms that have been developed to design metaheuristic implementations, both of which are, at the moment, scattered in the literature. In such a scenario, the use of automatic design and of the component-based view of metaheuristics will be predominant, allowing the possibility to easily create implementations that include components from *any* kind of metaheuristic.

Assisted optimization problems modeling. The last research direction is worth devoting efforts in the future is the creation of tools that assist users to model the optimization problem they want to solve in a simpler way. This is something rarely discussed in the literature of metaheuristics, where the optimization problem to be tackled is typically already well-defined. However, in many real-life situations, modeling the optimization problem is not straightforward. This is the case, for example, in those situations in which (i) there are complex dependencies among the solution variables, and/or (ii) the environment of the problem is dynamic and the position of the optimum changes over time. One way to automatize the modeling of the optimization problems is by using some of the open source natural language processing (NLP) packages, such as NLTK, PyTorch-NLP and OpenNLP, to create a tool that is capable of analyzing the raw data available on the problem and obtaining insights about its structure to produce a model that is meaningful for the user.

For several years, we have seen the integration of the different areas of AI to devise ever more powerful approaches. This has allowed us not only to solve a wide array of complex problems that we were not able to solve before, but also to simplify the execution of a number of tasks that are encompassed in their solution. The field of metaheuristics, despite the existence of some regressive trends, has also been part of this integration with other areas of AI. In our opinion, the outlook for the field of metaheuristics is very exciting and full of opportunities. Notably, the field keeps moving towards an increasing level of automation. This automation will not only allow users to be assisted in the description of new challenging optimization problems and in the automatic creation of high-performing algorithms to solve them, but also, very importantly, will help us to better understand the many different techniques that have already been proposed so that we can continue improving them.

Appendix A

Applying the IWD to the Traveling Salesman Problem

In this appendix, we illustrate one iteration of the intelligent water drops (IWD) metaheuristic applied to the traveling salesman problem.

A.1 Traveling salesman problem

In the traveling salesman problem (TSP), a set of n cities has to be visited by a salesman using the shortest possible route. After visiting each of the n cities once, the salesman has to return to his home city. In formal terms, we are searching for the Hamiltonian tour of minimal length in a fully connected graph.

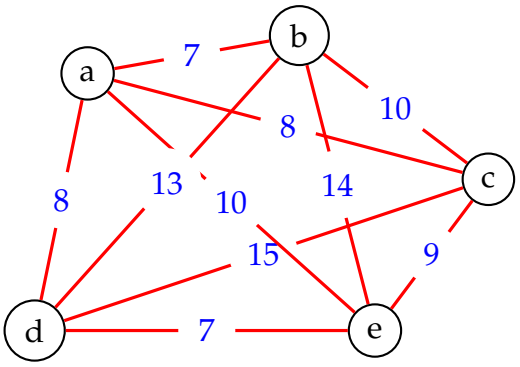


Figure A.1: A 5-cities graph for the TSP and distance between each city.

In Figure A.1 we show a graphical representation of the TSP instance considered in the example. The cities names are indicated within the graph nodes

and the distances d_{ij} between them is shown in color blue on the graph edges.

A.2 IWD notation and equations

The probability p_{ij}^k with which a water drop k placed on city i adds a non-yet-visited city j to the tour it is constructing is given by the *random selection rule*:

$$p_{ij}^k = \frac{\frac{1}{\epsilon + g(soil_{ij})}}{\sum_{ih \in N^f} \left(\frac{1}{\epsilon + g(soil_{ih})}\right)}, \tag{A.1}$$

where $soil_{ij}$ is the amount of soil on edge ij, ϵ is a small constant to avoid division by zero, N^f indicates the set of feasible solution components (i.e., edges that connect the current city to cities that have not been visited yet); therefore, $ih \in N^f$ indicates that city h can be added to a solution being constructed by a water drop placed on city i.

After a city j has been added to the tour constructed by water drop k:

- the amount of soil on the edge ($soil_{ij}$) is updated using the *local soil update* equation:

$$soil_{ij} = (1 - \varphi) \cdot soil_{ij} - \varphi \cdot \Delta soil_{ij}^k, \tag{A.2}$$

  where φ is a user selected parameter in the interval $[0, 1)$,

$$\Delta soil_{ij}^k = \frac{a_s}{b_s + c_s \cdot [HUD_{ij}/vel^k]^2}, \tag{A.3}$$

  and

$$vel^k = vel^k + \frac{a_v}{b_v + c_v \cdot [soil_{ij}]^2}, \tag{A.4}$$

- and the soil collected by the water drop k is updated using the equation:

$$collected\_soil^k = collected\_soil^k + \Delta soil_{ij}^k, \tag{A.5}$$

  where $\Delta soil_{ij}^k$ is defined as in Equation A.3

The parameters a_s, b_s and c_s in Equation A.3 and a_v, b_v and c_v in Equation A.4 are user selected parameters. HUD_{ij} is the heuristic undesirability of adding a city j to the tour while placed on city i, which, in this case, corresponds to the distance d_{ij} between cities i and j (see Fig. A.2).

$$HUD = \begin{pmatrix} 0 & 1 & 2 & 2 & 6 \\ 1 & 0 & 6 & 8 & 10 \\ 2 & 6 & 0 & 12 & 4 \\ 2 & 8 & 12 & 0 & 1 \\ 6 & 10 & 4 & 1 & 0 \end{pmatrix}$$

Figure A.2: Heuristic undesirability matrix

Once all solutions are completed, the water drop k^{best} that constructed the best tour s^{best} updates the values of $soil_{ij}$ on all the edges that belong to s^{best} using the *global soil update* rule:

$$soil_{ij} = \begin{cases} (1+\rho) \cdot soil_{ij} - \rho \cdot \Delta soil_{ij}^{best} & \text{if } ij \in s^{best} \\ soil_{ij} & \text{otherwise,} \end{cases} \tag{A.6}$$

where

$$\Delta soil_{ij}^{best} = collected\_soil^{best} / N - 1 \tag{A.7}$$

where N is the number of cities in the constructed solution, which in our example is 5.

In Table A.1 we give all the parameter values as recommended in (Shah-Hosseini 2009).

Table A.1: Recommended values for the IWD's parameters

Parameter:	a_v	b_v	c_v	a_s	b_s	c_s	$InitSoil$	$InitVel$	$collected\_soil$	φ	ρ
Value:	1	0.01	1	1	0.01	1	10000	200	0	0.9	0.9

A.3 Example

In this example, we make reference to the content of a spreadsheet file which is available as online supplementary material for the article (Camacho-Villalón et al. 2019) at: https://static-content.springer.com/esm/art%3A10.1007%2Fs11721-019-00165-y/MediaObjects/11721_2019_165_MOESM1_ESM.ods.

The algorithm to solve the TSP using IWD is shown in Algorithm 11. In this example, we set the number of water drops to 3 (i.e., $m = 3$). Let us

Algorithm 11 IWD algorithm for the traveling salesman problem

Output: Shortest tour found ST
Require:
$m \leftarrow 3$ ▷ number of water drops
$N \leftarrow 5$ ▷ number of cities
$ST \leftarrow \infty$ ▷ best solution found
$a_v \leftarrow 1$, $b_v \leftarrow 0.01$, $c_v \leftarrow 1$, $a_s \leftarrow 1$, $b_s \leftarrow 0.01$, $c_s \leftarrow 1$, $\varphi \leftarrow 0.9$, $\rho \leftarrow 0.9$
$s^{\text{best}} \leftarrow random\_solution$ ▷ initial iteration-best solution
for all (i, j) **do**
  $HUD \leftarrow d(i, j)$ ▷ initial heuristic undesirability values
  $soil_{ij} \leftarrow 10\,000$ ▷ initial soil values
end for
while (! TERMINATION()) **do**
  for $k \leftarrow 1$ **to** m **do** ▷ each water drop builds a solution
    $s^k \leftarrow ()$ ▷ initial empty partial solution
    $vel^k \leftarrow 200$ ▷ initial water drop velocity
    $collected\_soil^k \leftarrow 0$ ▷ initial collected soil
    select random initial city i ▷ select initial city
    $h \leftarrow i$
    for $n \leftarrow 2$ **to** N **do**
      select next city j ▷ use Equation A.1
      add city j to s^k ▷ add the city to the solution
      update $soil_{hj}$ ▷ use Equation A.2
      update $collected\_soil^k$ ▷ use Equation A.5
      $h \leftarrow j$
      if $(n = N)$ **then**
        add city i to s^k ▷ return to the initial city
        update $soil_{hi}$ ▷ use Equation A.2
        update $collected\_soil^k$ ▷ use Equation A.5
      end if
      $n \leftarrow n + 1$
    end for
    $k \leftarrow k + 1$
  end for
  for all k **do** ▷ iteration-best solution is computed
    if $\left(F(s^k) < F(s^{\text{best}})\right)$ **then** ▷ $F(\cdot)$ returns the cost of the solution
      $s^{\text{best}} \leftarrow s^{\text{k}}$
    end if
  end for
  for all $(i, j) \in s^{\text{best}}$ **do** ▷ global update
    update $soil_{ij}$ ▷ use Equation A.6
  end for
  if $\left(F(s^{\text{best}}) < F(ST)\right)$ **then**
    $ST \leftarrow s^{\text{best}}$ ▷ new global-best solution
  end if
  $s^{\text{best}} \leftarrow random\_solution$
end while
Return ST

suppose that in the first iteration of the algorithm the 3 water drops construct the following tours:

$$\text{k=1}: \textit{d-e-b-c-a-d}$$

$$\text{k=2}: \textit{a-e-d-c-b-a}$$

$$\text{k=3}: \textit{d-e-c-b-a-d}$$

that is, for k=1, *d* is the initial city randomly selected (line 10, Algorithm 11), and cities *e*, *b*, *c* and *a* are then selected by means of the *random selection rule* (line 13, Algorithm 11). Finally, the tour is completed by deterministically adding the inital city *d* (line 19, Algorithm 11). Similarly for $k = 2$ and for $k = 3$.

Using these tours, the rest of the execution of Algorithm 11 is described using the content of the spreadsheet file. First, the *HUD* matrix (line 2, Algorithm 11) is defined in cells A1:G6, and the initial soil matrix (line 3, Algorithm 11) is defined in cells A9:G14, The parameters of the algorithm are defined in cells I1:J8 (see also Table A.1). The user may modify the value of these parameters to try the example using different parameter configurations.

For every city added to a solution, we show in the spreadsheet file the computation of the water drops new velocity vel^k, of the variable $\Delta soil_j^k$, of the new value of $soil_{ij}$ (lines 15 and 20, Algorithm 11) and the new value of $collected\_soil^k$ (lines 16 and 21, Algorithm 11). The new values of $soil_{ij}$ are shown highlighted in the updated soil matrix shown on the right side (e.g., in the updated soil matrix of cells D22:H26, soil is updated in cells G26 and H25).

After the three water drops have completed their tours, we show the application of the global soil update in cells A153:C158, and the resulting soil matrix in cells E152:J158. In Algorithm 11 this is shown from lines 27 to 34.

Note that after the application of the global soil update and using the parameters recommended by the author (see Table A.1), some of the graph edges included in the best solution end up with a higher soil value. This is counter-intuitive as the edges of the best quality solution should rather decrease their value. In fact, as we explain in the article, this will happens as long as the value of $soil_{ij}$ is positive and the right-hand side of Equation A.6 has a smaller value than the left-hand side—i.e., $(collected\_soil^{\text{best}}/N_{\text{best}} - 1) \leq (1 + \rho) \cdot soil_{ij}$.

Appendix B

Supplementary Material for PSO-*X*

This appendix contains supplementary material for the PSO-*X* framework described in Chapter 8. In Section B.1, we present a number of tables that contain: (i) the abbreviations used in the Chapter 8 (Table B.1); (ii) the complete list of parameters of the PSO-*X* framework (Table B.2); (iii) the forbidden configurations when executing PSO-*X* (Table B.3); (iv) the available strategies for computing the main parameters of the generalized velocity update rule that we proposed (Table B.4); and (v) the parameter settings of the default and tuned versions of the ten PSO variants included in our comparison (Table B.5). In Section B.2, we present an analysis of the convergence of the PSO-*X* algorithms and of the PSO variants according to Poli (2009) and Poli and Broomhead (2007)' theoretical convergence bounds, as well as a number of plots that show the sampling distribution of the parameters and the way in which they interact with each other. The algorithms of the six PSO-*X* implementations and of the ten PSO variants are reported in Section B.3. Finally, we present the distribution of the median solution of the 16 compared algorithms using box-plots, in Section B.4; and the best solutions quality vs function evaluations plots, in Section B.5.

B.1 Parameter Settings

B.1.1 PSO-*X* Parameters

PSO-*X* has a total of 58 parameters. However, not all of them are used at once when the framework is executed. The actual number of parameters depends on the options selected for Population, DNPP, Topology, Model of influence, ω_1, ω_2 and ω_3. In Table B.2, we show the complete list of parameters of PSO-*X*, their

Table B.1: List of abbreviations used in Chapter 8

Abbreviations	
PSO	Particle swarm optimization
DNPP	Distribution of all next possible positions
GVUR	Generalized velocity update rule
CI	Cognitive influence
SI	Social influence
$\mathsf{Pert}_{\mathsf{rand}}$	Random perturbation
$\mathsf{Pert}_{\mathsf{info}}$	Informed perturbation
RRMs	Random rotation matrices
PM	Perturbation magnitude
Mtx	Matrix
Top	Topology
MoI	Model of influence
Pop	Population
ACs	Acceleration coefficients
CCVUR	Constriction coefficient velocity update rule
ERiPSO	Enhanced rotation invariant PSO variant
FiPSO	Fully informed PSO variant
FraPSO	Frankenstein's PSO variant
GauPSO	Gaussian "bare-bones" PSO variant
HiePSO	Hierarchical PSO variant
IncPSO	Incremental PSO variant
LcRPSO	Locally convergent rotation invariant PSO variant
ResPSO	Restart PSO variant
SPSO11	Standard PSO 2011 variant
StaPSO	Standard particle swarm optimization

type and domain, and the specific condition(s) under which each parameter has to be configured. Note that, in Table B.2, we use the parameters names as defined in the code of PSO-*X*; these are the names that users have to specify as command line arguments when executing the framework. To avoid confusion, we also show (in parenthesis) the names of the parameters as they are presented and described in Chapter 8.

Table B.2: Complete list of parameters as they have to be indicated when executing PSO-*X*. The names in parenthesis in the first column are the ones used in Chapter 8.

Name	Type	Domain	Condition(s)
Population			
populationCS (Pop)	categorical	0=Pop-constant, 1=Pop-time-varying, 2=Pop-incremental	
particles	integer	(2, 200)	populationCS $\in \{0, 1\}$
initialPopSize ($\texttt{pop}_{ini}$)	integer	(2, 10)	populationCS $\in \{1, 2\}$
finalPopSize ($\texttt{pop}_{fin}$)	integer	(2, 200)	populationCS $\in \{1, 2\}$
particlesToAdd (ξ)	integer	(1, 10)	populationCS = 2
pIntitType (Init)	integer	0=Init-random, 1=Init-horizontal	populationCS = 2
popTViterations (k)	integer	(1, 100)	populationCS = 1
DNPP			
DNPP	categorical	0=DNPP-rectangular, 1=DNPP-spherical, 2=operator_q	
operator_q ($\vec{q}$)	categorical	0=DNPP-standard, 1=DNPP-Gaussian, 2=DNPP-discrete, 3=DNPP-Cauchy–Gaussian	DNPP = 2
randNeighbor (MoI-random informant)	categorical	0=false, 1=true	DNPP = 2
operatorCG_parm_r (r)	real	(0.00, 1.00)	operator_q = 3
Topology			
topology (Top)	categorical	0=Top-ring, 1=Top-fully-connected, 2=Top-wheel, 3=Top-Von Neumann, 4=Top-random edge, 5=Top-hierarchical, 6=Top-time-varying	
modInfluence (MoI)	categorical	0=MoI-best-of-neighborhood, 1=MoI-fully informed, 2=MoI-ranked fully informed	
branching (bd)	integer	(2, 20)	topology = 6

Table B.2 Continued.

Name	Type	Domain	Condition(s)
Acceleration coefficients			
accelCoeffCS (AC)	categorical	0=AC-constant, 1=AC-random, 2=AC-time-varying, 3=AC-extrapolated	DNPP $\in \{0, 1\} \vee$ ((DNPP = 2) $\wedge$ (operator_q = 0))
phi1 (φ_1)	real	(0.00, 2.50)	accelCoeffCS = 0
phi2 (φ_2)	real	(0.00, 2.50)	accelCoeffCS = 0
initialPhi1 (φ_{1min})	real	(0.00, 2.50)	accelCoeffCS $\in \{1, 3\}$
finalPhi1 (φ_{1max}	real	(0.00, 2.50)	accelCoeffCS $\in \{1, 3\}$
initialPhi2 (φ_{2min})	real	(0.00, 2.50)	accelCoeffCS $\in \{1, 3\}$
finalPhi2 ((φ_{2max})	real	(0.00, 2.50)	accelCoeffCS $\in \{1, 3\}$
***Random* perturbation**			
perturbation2 ($\text{Pert}_{\text{rand}}$)	categorical	0=none, 1=$\text{Pert}_{\text{rand}}$-rectangular, 2=$\text{Pert}_{\text{rand}}$-noisy	omega2CS $\in \{0, 2, 3, 4\}$
magnitude2CS (PM)	categorical	1=PM-constant value, 2=PM-Euclidean distance, 3=PM-obj.func. distance, 4=PM-success rate	perturbation2 $\in \{1, 2\}$
magnitude2 ($\text{PM}_{t=0}$)	real	(0.0, 1.0)	magnitude2CS $\in \{1, 4\}$
mag2_parm_l_CS	categorical	0=variable, 1=constant	magnitude2CS = 2
mag2_par_l (ϵ)	real	(0.0, 1.0)	mag2_parm_l_CS = 1
mag2_par_m (m)	real	(0.0, 1.0)	magnitude2CS = 3
mag2_parm_success (s_c)	integer	(1, 50)	magnitude2CS = 4
mag2_parm_failure (f_c)	integer	(1, 50)	magnitude2CS = 4
Inertia			

Table B.2 Continued.

Name	Type	Domain	Condition(s)
omega1CS (ω_1)	categorical	0=constant 1=linear decreasing, 2=linear increasing, 3=random, 11=self-regulating, 12=adaptive based on velocity, 13=double exponential self-adaptive, 14=rank-based, 15=success-based, 16=convergence-based,	

In Table B.3, we show the list of forbidden configurations when executing PSO-X, i.e., parameter values or combinations of parameter values that are not allowed. When a new configuration is created, `irace` evaluates whether the configuration is valid based on the forbidden configurations file and automatically discard it if it matches any of the entries in the file.

Table B.3: List of forbidden configurations when executing PSO-X.

Forbidden configurations
initialPopSize $>$ finalPopSize
(topology = 6) $\wedge$ (modInfluence = 2)
(branching $>$ finalPopSize) $\vee$ (branching $>$ particles) $\vee$ (branching $<$ initialPopSize)
(DNPP = 2) $\wedge$ randomMatrix $\in \{1,2,3,4,5,6\}$
particlesToAdd $>$ (finalPopSize$-$initialPopSize)
(accelCoeffCS = 1) $\wedge$ (initialPhi1 $\leq$ finalPhi1)
(accelCoeffCS = 1) $\wedge$ (initialPhi2 $\geq$ finalPhi2)
(accelCoeffCS = 3) $\wedge$ (finalPhi1 $\leq$ initialPhi1)
(randNeighbor = 1) $\wedge$ modInfluence $\in \{0,2\}$
(populationCS = 1) $\wedge$ ((particles $<$ initialPopSize) $\vee$ (particles $>$ finalPopSize))
accelCoeffCS $\in \{1,2,3\}$ $\wedge$ (operator_q $\in \{1,2,3)$
(unstuck = 1) $\wedge$ (inertia = 0.0)

Finally, in Table B.4, we show the strategies available for computing parameters ω_1, ω_2 and ω_3 of the generalized velocity update rule (GVUR).

Table B.4: Available strategies for computing ω_1, ω_2 and ω_3 and their mathematical definition.

Parameter	Strategy	Mathematical definition
ω_1*	constant	self-explanatory
	linear decreasing	$\omega_{1t} = \omega_{1max} - (\omega_{1max} - \omega_{1min})\frac{t}{t_{max}}$
	linear increasing	$\omega_{1t} = \omega_{1min} - (\omega_{1min} - \omega_{1max})\frac{t}{t_{max}}$
	random	$\omega_{1t} \sim \mathcal{U}[0.5, 1)$
	self-regulating	$\omega_{1t}^i = \begin{cases} \omega_{1t-1}^i + \eta\Delta\omega & \text{if } \vec{p}_t^{\,i} = \vec{g}_t \\ \omega_{1t-1}^i - \Delta\omega & \text{otherwise} \end{cases}$, where $\Delta\omega = \frac{(\omega_{1max} - \omega_{1min})}{t_{max}}$ and η is a user selected parameter
	adaptive based on velocity	$\omega_{1t} = \begin{cases} \arg\max\{\omega_{1t-1} - \lambda, \omega_{1min}\} & \text{if } \overline{v_t} \geq v_{t+1}^{\text{ideal}} \\ \arg\min\{\omega_{1t-1} + \lambda, \omega_{1max}\} & \text{otherwise} \end{cases}$, where λ is a user selected parameter, $\overline{v_t} = \frac{1}{n \cdot d}\sum_{i=1}^{n}\sum_{j=1}^{d}\lvert v_t^{i,j}\rvert$, $v_t^{\text{ideal}} = v_s\left(\frac{1+\cos\left(\pi\frac{t}{0.95 \cdot t_{max}}\right)}{2}\right)$ and $v_s = \frac{ub-lb}{2}$
	double exponential self-adaptive	$\omega_{1t}^i = e^{-e^{-R_t^i}}$, where $R_t^i = \lvert\lvert\vec{p}_t^{\,\text{best}} - \vec{x}_t^{\,i}\rvert\rvert(\frac{t_{max}-t}{t_{max}})$
	rank-based	$\omega_{1t} = \omega_{1min} + (\omega_{1max} - \omega_{1min})\frac{\mathcal{R}_t(i)}{n}$, where $\mathcal{R}_t(i)$ is a function that returns the fitness rank of i
	success-based	$\omega_{1t} = \omega_{1min} + (\omega_{1max} - \omega_{1min})\frac{\sum_{i=1}^{n} S_t^i}{n}$, where $S_t^i = \begin{cases} 1 & \text{if } f(\vec{p}_t^{\,i}) < f(\vec{p}_{t-1}^{\,i}) \\ 0 & \text{otherwise} \end{cases}$
	convergence-based	$\omega_{1t}^i = 1 - \left\lvert\frac{a - C_t^i}{(1+D_t^i)(1+b)}\right\rvert$, where $a, b \in [0, 1]$ are user selected parameters, $C_t^i = \frac{\lvert f(\vec{p}_{t-1}^{\,i}) - f(\vec{p}_t^{\,i})\rvert}{f(\vec{p}_{t-1}^{\,i}) - f(\vec{p}_t^{\,i})}$ and $D_t^i = \frac{\lvert f(\vec{p}_t^{\,i}) - f(\vec{l}_t^{\,i})\rvert}{f(\vec{p}_t^{\,i}) - f(\vec{l}_t^{\,i})}$
ω_2, ω_3**	$= \omega_1$	$\omega_2 = \omega_1$, $\omega_3 = \omega_1$
	random	$\omega_{2t}, \omega_{3t} \sim \mathcal{U}[0.5, 1]$
	constant	ω_2, ω_3 are user selected constants in the range $[0, 1]$

* In the strategies for computing ω_1, t_{max} indicate the maximum number of iterations set for the algorithm and ω_{1min}, ω_{1max} are user selected constants for the minimum and maximum allowable value of ω_1.

** Note that different combinations are possible, for example $\omega_2 = \omega_1$ and $\omega_3 =$ random.

B.1.2 PSO Variants Parameters

In Table B.5, we show the parameter setting of the default and tuned version of the ten PSO variants included in the comparison with the automatically generated PSO-*X* algorithms.

Table B.5: Parameter settings of the default (*dft*) and tuned (*tnd*) version of the ten PSO variants included in the comparison.

Algorithm		Settings
ERiPSO	*dft*	pop $= 20$, $\omega_1 = 0.7213475$, AC-random, $\varphi_{1min} = 0$, $\varphi_{1max} = 2.05$, $\varphi_{2min} = 0$, $\varphi_{2max} = 2.05$, Mtx-Euclidean rotation$_{all}$ with α-adaptive and $\zeta = 30$ and $\rho = 0.01$.
	tnd	Top-Von Neumann, pop $= 166$, $\omega_1 = 0.4746$, AC-random, $\varphi_{1min} = 0.825$, $\varphi_{1max} = 1.686$, $\varphi_{2min} = 1.4733$, $\varphi_{2max} = 2.1776$, Mtx-Euclidean rotation$_{one}$ with α-Gaussian and $\sigma = 39.8385$.
FiPSO	*dft*	pop$= 20$, $\omega_1 = \omega_2 = 0.729843788$, Mtx-random diagonal, $\varphi_1 = 2.05$ and $\varphi_2 = 2.05$.
	tnd	Top-ring, pop $= 20$, $\omega_1 = \omega_2 = 0.729843788$, Mtx-random diagonal, $\varphi_1 = 2.1864$ and $\varphi_2 = 2.3156$.
FraPSO	*dft*	$\kappa = 60$, pop $= 60$, $\omega_1 =$ linear decreasing, $\omega_{1min} = 0.4$, $\omega_{1max} = 0.9$, $t_{schd} = 600$, $\varphi_1 = 2.0$ and $\varphi_2 = 2.0$.
	tnd	$\kappa = 300$, pop $= 30$, $\omega_1 =$ linear decreasing, $\omega_{1min} = 0.0022$, $\omega_{1max} = 0.8625$, $t_{schd} = 210$, $\varphi_1 = 2.0322$ and $\varphi_2 = 1.9605$.
GauPSO	*dft*	pop $= 20$, $\omega_1 = 0$ and DNPP-additive stochastic DNPP-Gaussian.
	tnd	Top-time-varying with Mol-random informant, $\kappa = 150$ pop $= 30$, $\omega_1 = 0$ and DNPP-additive stochastic DNPP-Gaussian.
HiePSO	*dft*	$bd = 5$, pop $= 40$, $\omega_1 =$ linear increasing, $\omega_{1min} = 0.4$, $\omega_{1max} = 0.9$, $\varphi_1 = 1.496180$ and $\varphi_2 = 1.496180$.

* ζ and ρ are parameters of α-adaptive; σ is a parameter of α-Gaussian; κ is a parameter of Top-time-varying; t_{schd} is a parameter of $\omega_1 =$ linear decreasing; bd is a parameter of Top-hierarchical; ξ is a parameter of Pop-incremental; and ϵ is a parameter of PM-Euclidean distance.

Table B.5 Continued.

Algorithm		Settings
HiePSO	*tnd*	$bd = 2$, pop $= 114$, $\omega_1 =$ linear increasing, $\omega_{1min} = 0.3284$, $\omega_{1max} = 0.8791$, $\varphi_1 = 2.1105$ and $\varphi_2 = 1.0349$.
IncPSO	*dft*	Init-horizontal, $\text{pop}_{ini} = 2$, $\text{pop}_{fin} = 1000$, $\xi = 1$, $\omega_1 = \omega_2 = 0.729843788$, $\varphi_1 = 2.05$ and $\varphi_2 = 2.05$.
	tnd	Top-time-varying, $\kappa = 2360$, Init-horizontal, $\text{pop}_{ini} = 5$, $\text{pop}_{fin} = 295$, $\xi = 10$, $\omega_1 = \omega_2 = 0.729843788$, $\varphi_1 = 1.9226$ and $\varphi_2 = 1.0582$.
LcRPSO	*dft*	pop $= d$, $\omega_1 = 0.7298$, AC-random, $\varphi_{1min} = 0$, $\varphi_{1max} = 1.4962$, $\varphi_{2min} = 0$, $\varphi_{2max} = 1.4962$, Mtx-random linear, $\text{Pert}_{\text{info}}$-Gaussian with PM-Euclidean distance and $\epsilon = 0.46461/d^{0.58}$.
	tnd	Top-Von Neumann, pop $= d$, $\omega_1 = 0.1306$, AC-random, $\varphi_{1min} = 0.3142$, $\varphi_{1max} = 0.9002$, $\varphi_{2min} = 0.3228$, $\varphi_{2max} = 1.5209$, Mtx-random linear, $\text{Pert}_{\text{info}}$-Gaussian with PM-Euclidean distance and $\epsilon = 0.249$.
ResPSO	*dft*	pop $= 10$, $\omega_1 =$ linear decreasing, $\omega_{1min} = 0$, $\omega_{1max} = 0.1$, $\varphi_1 = 1.5$, $\varphi_2 = 1.5$.
	tnd	Top-ring with MoI-fully informed, pop $= 10$, $\omega_1 =$ linear decreasing, $\omega_{1min} = 0.2062$, $\omega_{1max} = 0.6446$, $\varphi_1 = 1.5014$, $\varphi_2 = 2.2955$.
SPSO11	*dft*	Top-ring, pop $= 40$, $\omega_1 = 0.7213475$, $\varphi_1 = 1.193147$, $\varphi_2 = 1.193147$.
	tnd	Top-time-varying with $\kappa = 1085$, pop $= 155$, $\omega_1 = 0.6482$, $\varphi_1 = 2.2776$, $\varphi_2 = 2.1222$.
StaPSO	*dft*	Top-fully-connected, pop $= 40$, $\omega_1 = 0.7298$, $\varphi_1 = 1.496180$, $\varphi_2 = 1.496180$.
	tnd	Top-Von Neumann, pop $= 34$, $\omega_1 = 0.6615$, $\varphi_1 = 2.3706$, $\varphi_2 = 0.8914$.

* ζ and ρ are parameters of α-adaptive; σ is a parameter of α-Gaussian; κ is a parameter of Top-time-varying; t_{schd} is a parameter of $\omega_1 =$ linear decreasing; bd is a parameter of Top-hierarchical; ξ is a parameter of Pop-incremental; and ϵ is a parameter of PM-Euclidean distance.

B.2 Analysis of Convergence and Parameters Interaction

B.2.1 PSO-*X* Algorithms Convergence

Local convergence is one of the most important characteristics of high-performing PSO implementations; it prevents issues such as *swarm explosion* and allows particles to improve their initial solutions for any number of dimensions. According to Poli (2009) and Poli and Broomhead (2007), PSO implementations whose inertia weight values (ω_1, in our case) is limited in the region defined by

$$\frac{5\,C - g(C)}{48} < \omega_1 < \frac{5\,C + g(C)}{48}, \tag{B.1}$$

where

$$C = \varphi_1 + \varphi_1$$

and

$$g(C) = \sqrt{25\,C^2 - 672\,C + 2304},$$

are expected to exhibit a locally convergent behavior.

Using Equation B.1, we computed the lower and upper bounds of ω_1 for the six PSO-*X* implementations and the ten PSO variants. In the case of PSO-X_{all}, which uses the convergence-based strategy (see Table B.4), the range of ω_1 can be obtained by substituting $a = 0.7192$, $b = 0.9051$ in the following equation:

$$\omega_{1t}^{i} = 1 - \left| \frac{a - C_t^i}{(1 + D_t^i)(1 + b)} \right| \tag{B.2}$$

and computing its limit when $C_t^i \to 0 \wedge D_t^i \to 0 = 0.6224$, $C_t^i \to 0 \wedge D_t^i \to 1 = 0.8112$, $C_t^i \to 1 \wedge D_t^i \to 0 = 0.8526$ and $C_t^i \to 1 \wedge D_t^i \to 1 = 0.9263$, which results in $\omega_{1min} = 0.6224$ and $\omega_{1max} = 0.9263$.

In the case of PSO-X_{hyb}, PSO-X_{uni} and PSO-X_{cec}, where parameters φ_1 and φ_2 are obtained by sampling from a random uniform distribution, we consider the expected value of these parameters, that is, $\varphi_1 = E[\varphi_{1min}, \varphi_{1max}]$ and $\varphi_2 = E[\varphi_{2min}, \varphi_{2max}]$. Finally, in the case of PSO-X_{hyb}, PSO-X_{uni}, PSO-X_{cec} and PSO-X_{soco}, we take into account the influence of parameter ω_2, which is a value between 0 and 1 that multiplies the acceleration coefficients in the same way as the constriction coefficient does in PSO implementations using the constriction

coefficient ($\chi = 0.7298$)(Clerc and Kennedy 2002). The results of our analysis of convergence are shown in Table B.6 and Table B.7.

Table B.6: Value of the main control parameters of the PSO-X algorithms and the lower and upper bounds for order-1 stability according to Poli's theoretical analysis.

Algorithm	φ_1	φ_2	Poli's lower bound	ω_1	Poli's upper bound
PSO-X_{all}	1.7067	2.2144	-0.1795	(**0.6224**, 0.9263)	0.7922
PSO-X_{hyb}	1.6041	1.6852	-0.0548	(**0.1190**, **0.1378**)	0.7401
PSO-X_{mul}	0.9200	1.6577	-0.2974	(**0.4000**, 0.9000)	0.8344
PSO-X_{uni} *	1.4217	2.0510	$(-0.7377, -0.4537)$	(**0.3531**, **0.7095**)	$(0.8821, 0.9509)$
PSO-X_{cec}	1.8958	0.8972	-0.4425	(**0.1673**, **0.2317**)	0.8789
PSO-X_{soco} *	0.7542	1.9235	$(-0.5386, -0.4129)$	(**0.6564**, **0.8201**)	$(0.8704, 0.9048)$

* In these algorithms, the value of the acceleration coefficients is constricted by the range of values that ω_1 can take, which results in two ranges (one for Poli's lower bound and one for the upper one) whose maximum and minimum values depend on ω_{1min} and ω_{1max}.

As it can be seen in Table B.6, with the exception of PSO-X_{all} and PSO-X_{mul}, the value of the ω_1 used by the PSO-X implementations is inside Poli's theoretical convergence region. This shows that the automatic design of PSO-X with `irace` produces implementations that are, overall, locally convergent. Although, in PSO-X_{all}, the value of ω_1 can exceed Poli's upper bound, it is worth noticing that large values of ω_1 are only expected in this algorithm at the beginning of its execution. This is because in order to obtain a $\omega_1 \sim \omega_{1max}$, it should be the case that the distance (measure in terms of solution quality) between a particle and the global best solution is very large, which may happen only during the initial iterations of the algorithm. In fact, as the execution of the algorithm progresses, the value of C_t^i tends to zero and the algorithm ends up exhibiting a convergent behavior. A similar situation to that of PSO-X_{all} occurs in the case of PSO-X_{mul}, where large ω_1 values are obtained only when particles succeed often in improving their personal best position. Since it becomes harder for particles to achieve a high success rate once they have moved to high quality areas of the search spaces, it is expected that the value of ω_1 decreases over time and the algorithm exhibits a locally convergent behavior.

Data from Table B.7 shows that the majority of the PSO variants in the comparison are inside Poli's theoretical convergence region, except for SPSO11$_{tnd}$, whose acceleration coefficient values are so large that computing Equation B.1

Table B.7: Value of the main control parameters of the PSO variants and the lower and upper bounds for order-1 stability according to Poli's theoretical analysis.

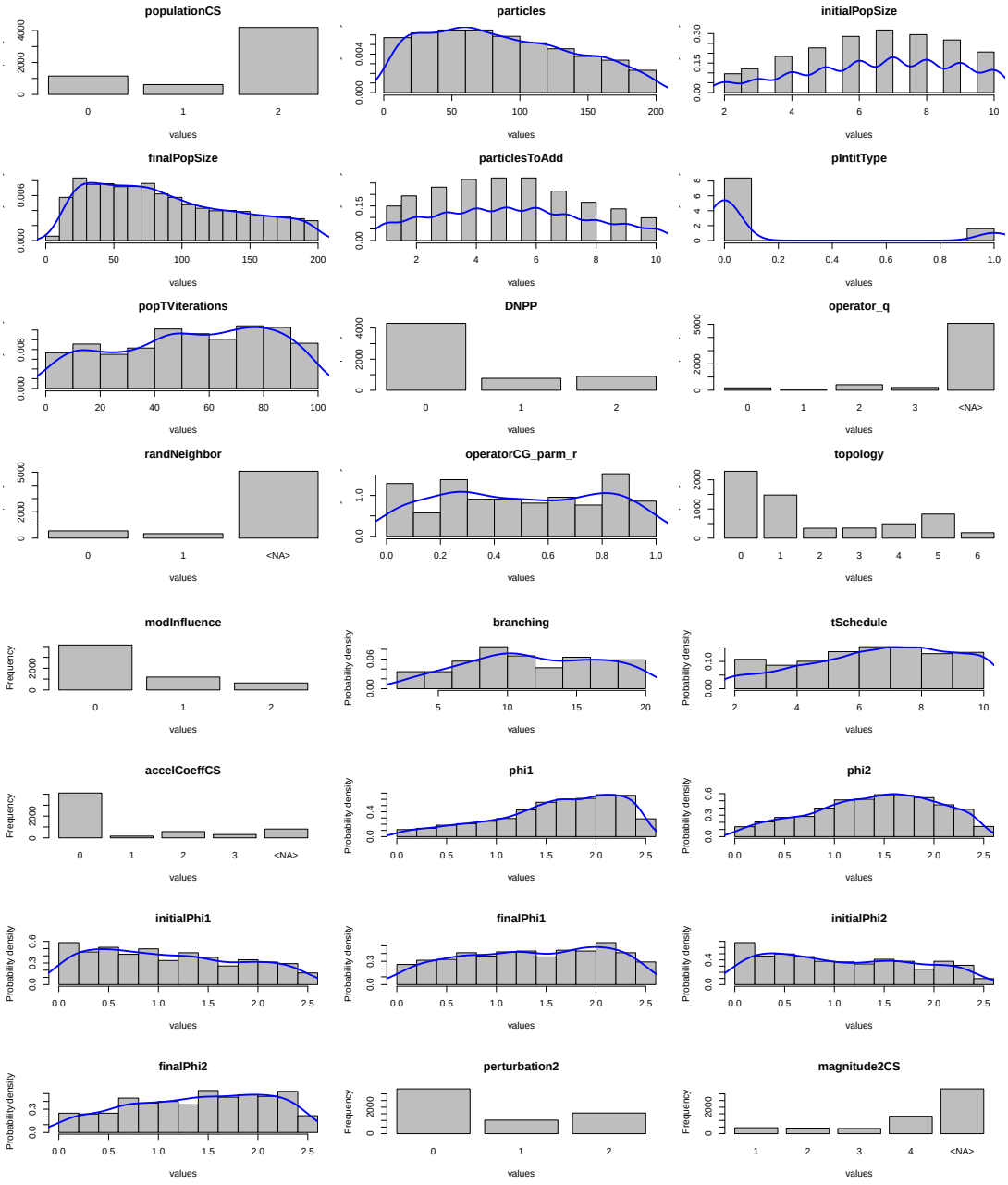
Algorithm	φ_1	φ_2	Poli's lower bound	ω_1	Poli's upper bound
ERiPSO$_{dft}$	1.0250	1.0250	-0.4555	**0.7213**	0.8826
FiPSO$_{tnd}$	2.1864	2.3156	-0.0563	**0.7298**	0.7407
FraPSO$_{dft}$	2.0000	2.0000	-0.1868	(**0.4000**, 0.9000)	0.7950
HiePSO$_{tnd}$	2.1105	1.0349	-0.1081	(**0.3284**, 0.8791)	0.7634
IncPSO$_{tnd}$	1.9226	1.0582	-0.4190	**0.7298**	0.8722
LcRPSO$_{dft}$	1.4962	1.4962	-0.1619	**0.7298**	0.7853
ResPSO$_{tnd}$	1.5014	2.2955	0.1741	(**0.2062**, 0.6446)	0.6168
SPSO11$_{tnd}$	2.2776	2.1222	—	0.6482	—
StaPSO$_{tnd}$	2.3706	0.8914	-0.0652	**0.6615**	0.7448

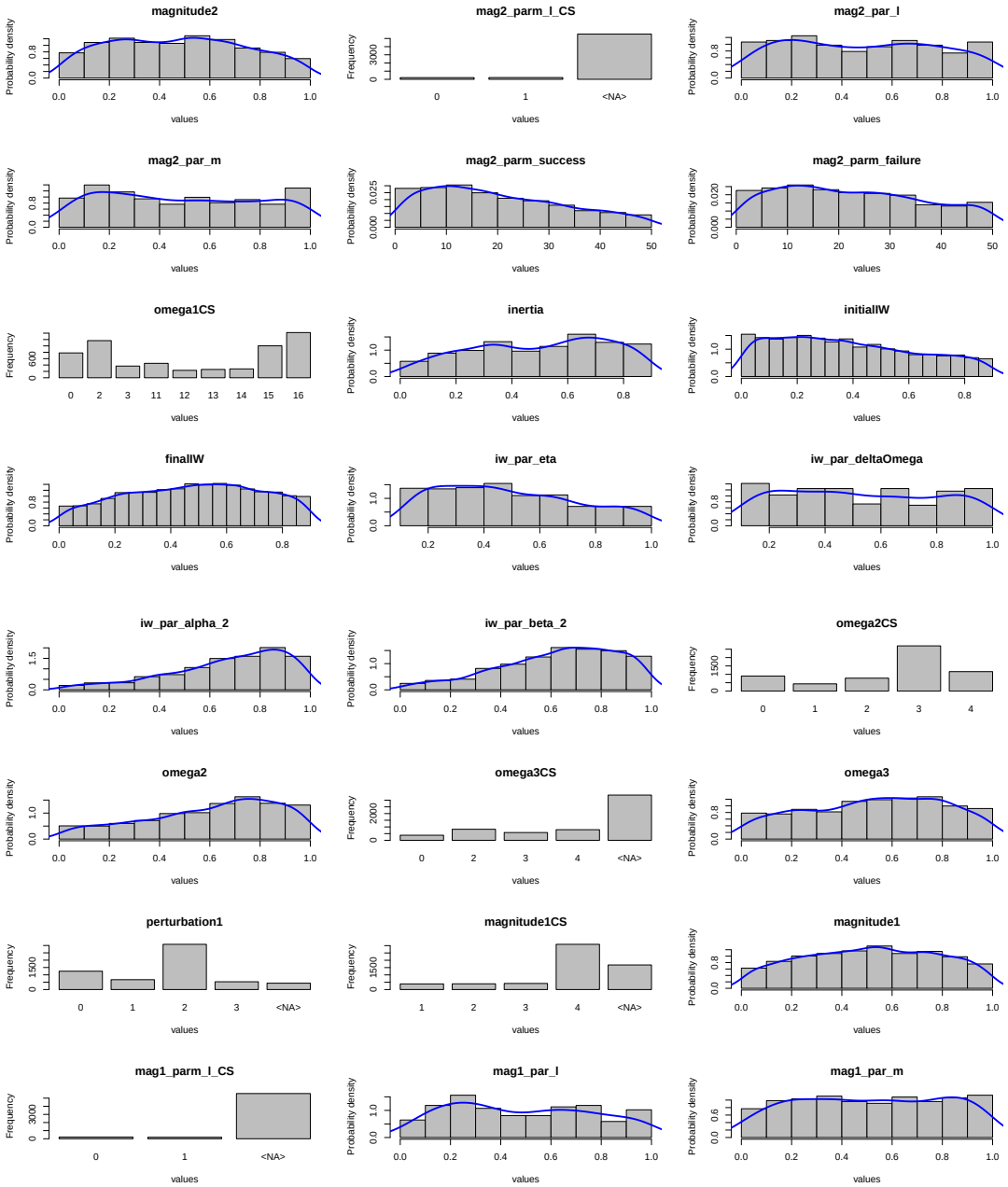
returned imaginary bounds. In the case of FraPSO$_{dft}$, HiePSO$_{tnd}$ and ResPSO$_{tnd}$, since the value of ω_1 is linearly decreasing, it is only during the initial iterations of the algorithms' execution that the upper convergence bound may be exceeded. In particular, in the case of ResPSO$_{tnd}$, the algorithm should remain inside the convergence bound for most of its execution time, since the difference between ω_{1max} and the computed Poli's upper bound is small. It is not clear why the configuration process with `irace` returned such large values for the acceleration coefficients of SPSO11$_{tnd}$; however, according to the results of our experiments, even with these parameter settings, the tuned version of SPSO11 was outperforming the default one.

B.2.2 Parameter Interaction

Given the large number of parameters in PSO-*X* compared to most PSO variants available in the literature, framework users could be interested in obtaining information about the sampling distribution of the parameters and the way in which they interact with each other. We present this information using the data gathered from the configuration process with `irace`. For each of the six PSO-*X* algorithms, we present the histograms of the sampled configurations and a parallel coordinate plot that depicts the interaction among the parameters during the last iterations of the iterated racing process (see Figs. 1 to 12).

PSO-X_{all}


populationCS
particles
initialPopSize
finalPopSize
particlesToAdd
pIntitType
popTViterations
DNPP
operator_q
randNeighbor
operatorCG_parm_r
topology
modInfluence
branching
tSchedule
accelCoeffCS
phi1
phi2
initialPhi1
finalPhi1
initialPhi2
finalPhi2
perturbation2
magnitude2CS
values
Frequency
Probability density


magnitude2
mag2_parm_l_CS
mag2_par_l
mag2_par_m
mag2_parm_success
mag2_parm_failure
omega1CS
inertia
initialIW
finalIW
iw_par_eta
iw_par_deltaOmega
iw_par_alpha_2
iw_par_beta_2
omega2CS
omega2
omega3CS
omega3
perturbation1
magnitude1CS
magnitude1
mag1_parm_l_CS
mag1_par_l
mag1_par_m
Probability density
Frequency
values
<NA>

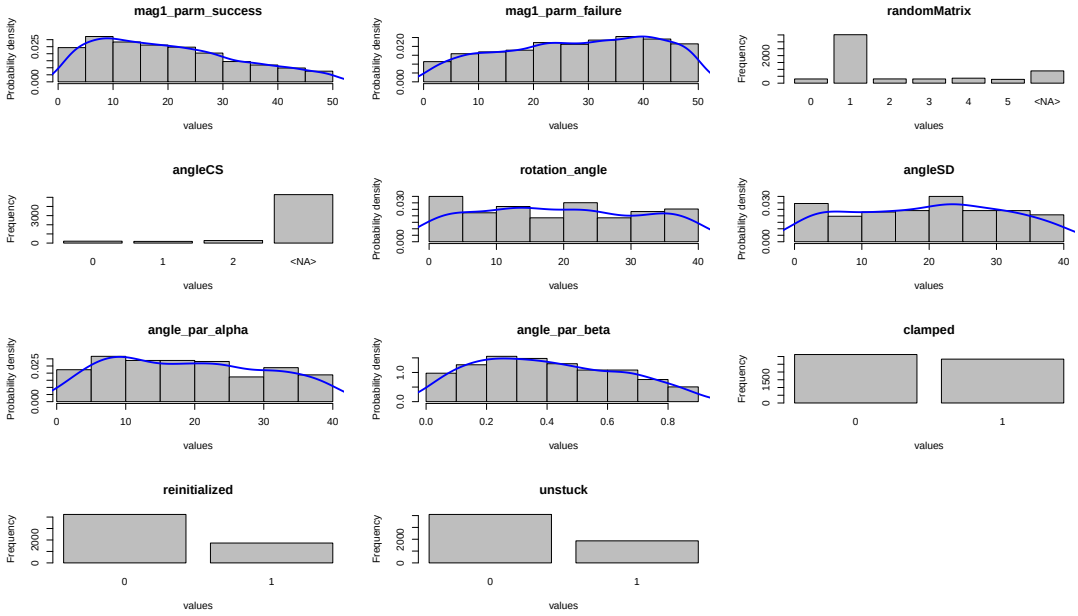


Figure B.1: Frequency of the sampled configurations of PSO-X_{all}

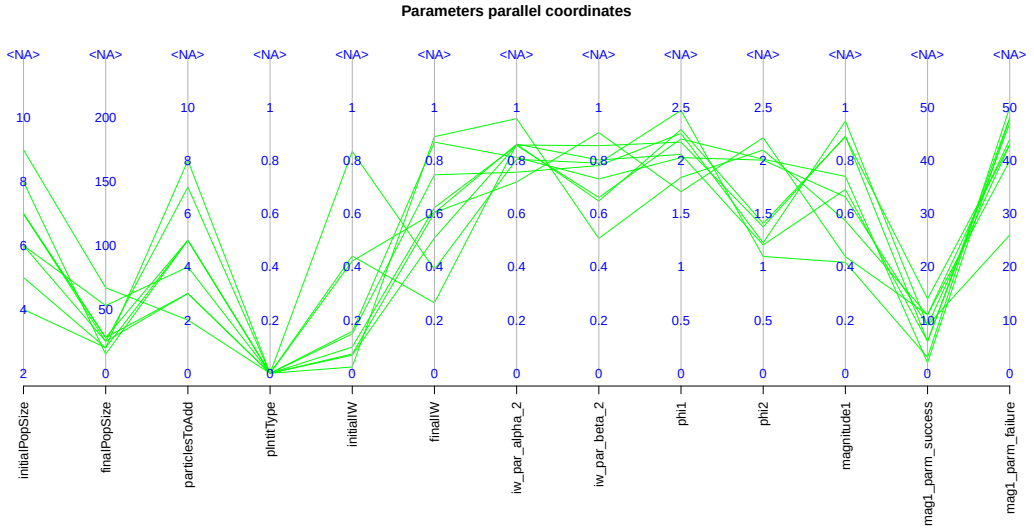
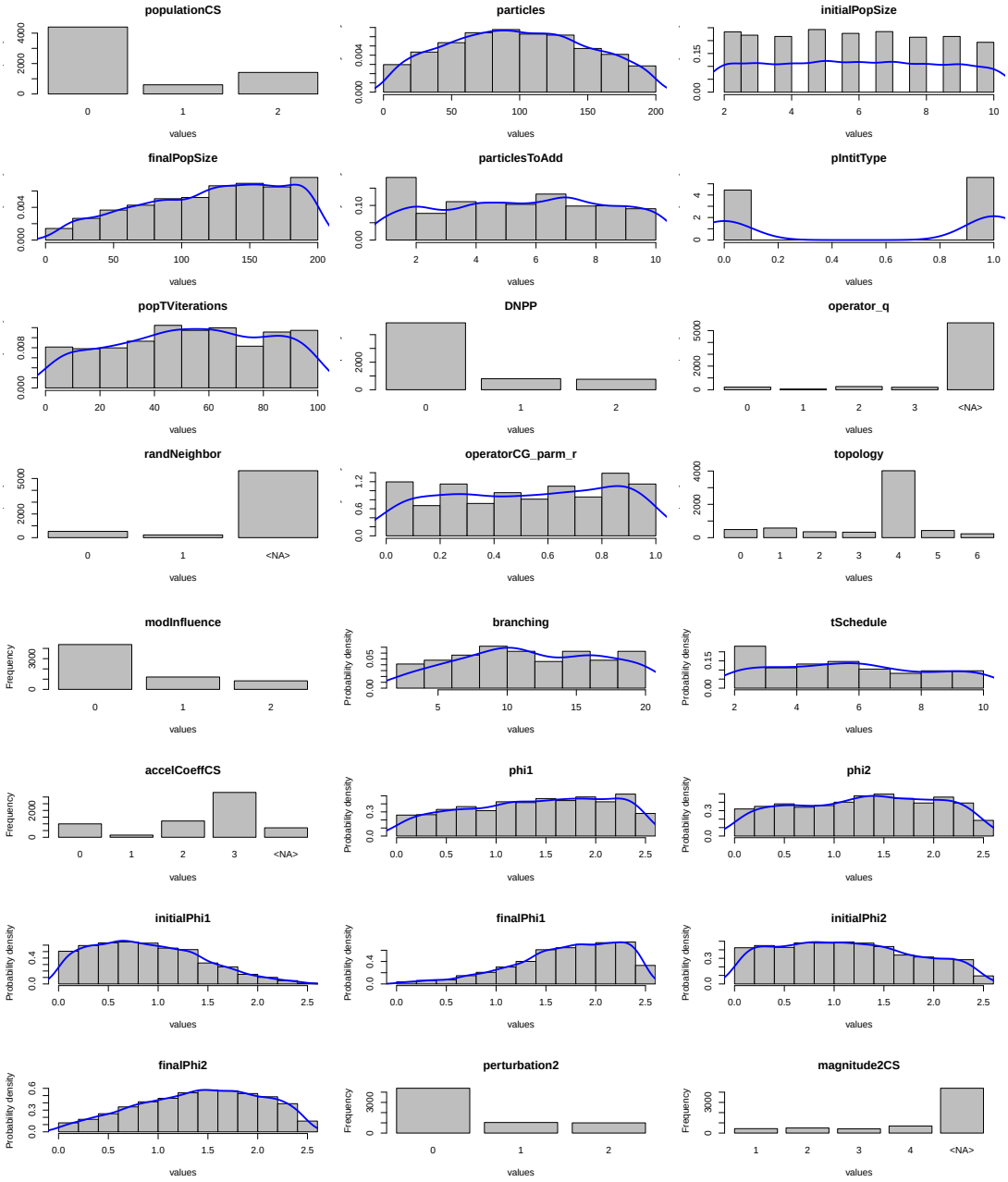
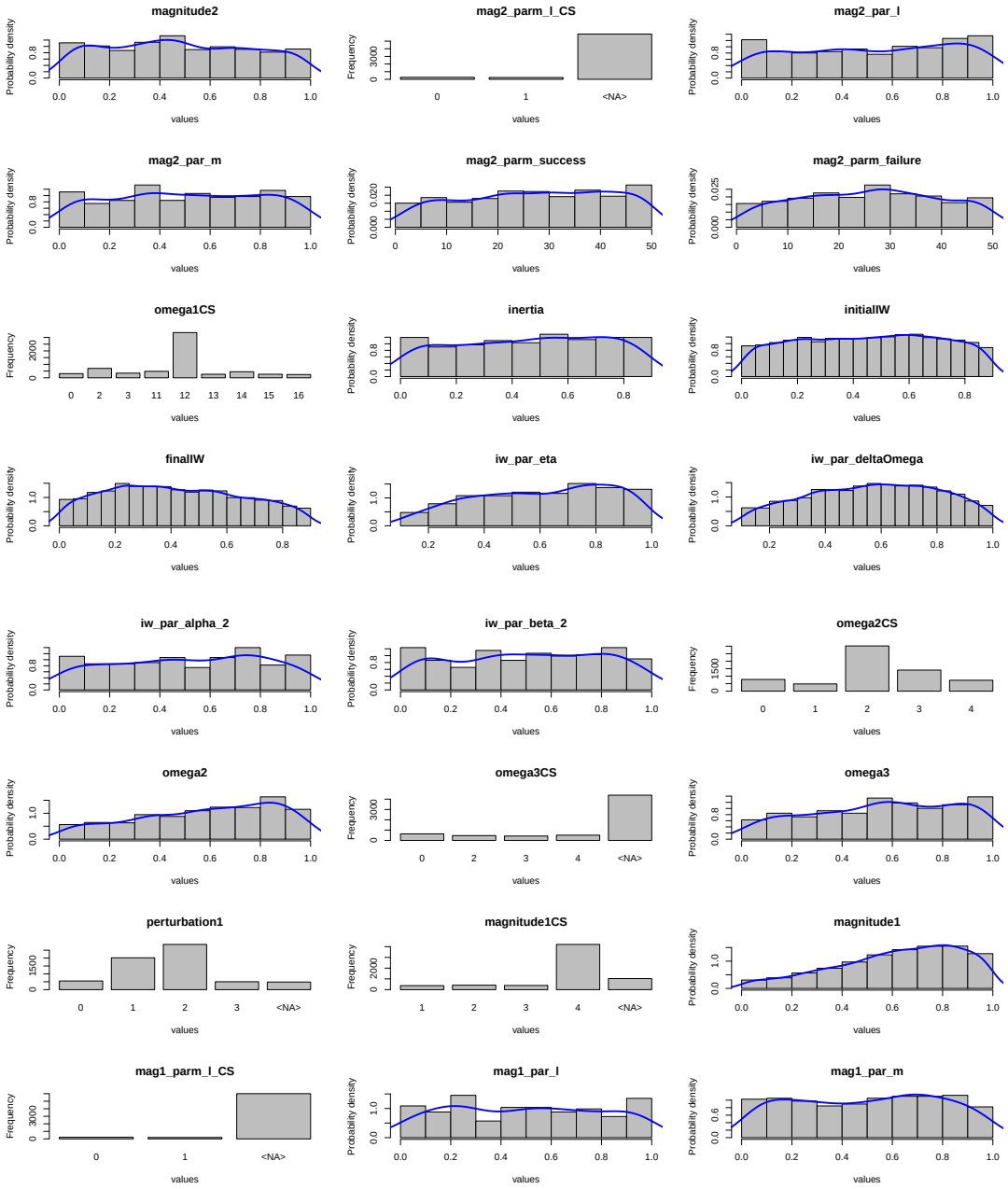


Figure B.2: Parameters interaction of PSO-X_{all}

PSO-X_{hyb}


populationCS
particles
initialPopSize
finalPopSize
particlesToAdd
pIntitType
popTViterations
DNPP
operator_q
randNeighbor
operatorCG_parm_r
topology
modInfluence
branching
tSchedule
accelCoeffCS
phi1
phi2
initialPhi1
finalPhi1
initialPhi2
finalPhi2
perturbation2
magnitude2CS
values
Frequency
Probability density


magnitude2
mag2_parm_l_CS
mag2_par_l
mag2_par_m
mag2_parm_success
mag2_parm_failure
omega1CS
inertia
initialIW
finalIW
iw_par_eta
iw_par_deltaOmega
iw_par_alpha_2
iw_par_beta_2
omega2CS
omega2
omega3CS
omega3
perturbation1
magnitude1CS
magnitude1
mag1_parm_l_CS
mag1_par_l
mag1_par_m
Probability density
Frequency
values
<NA>

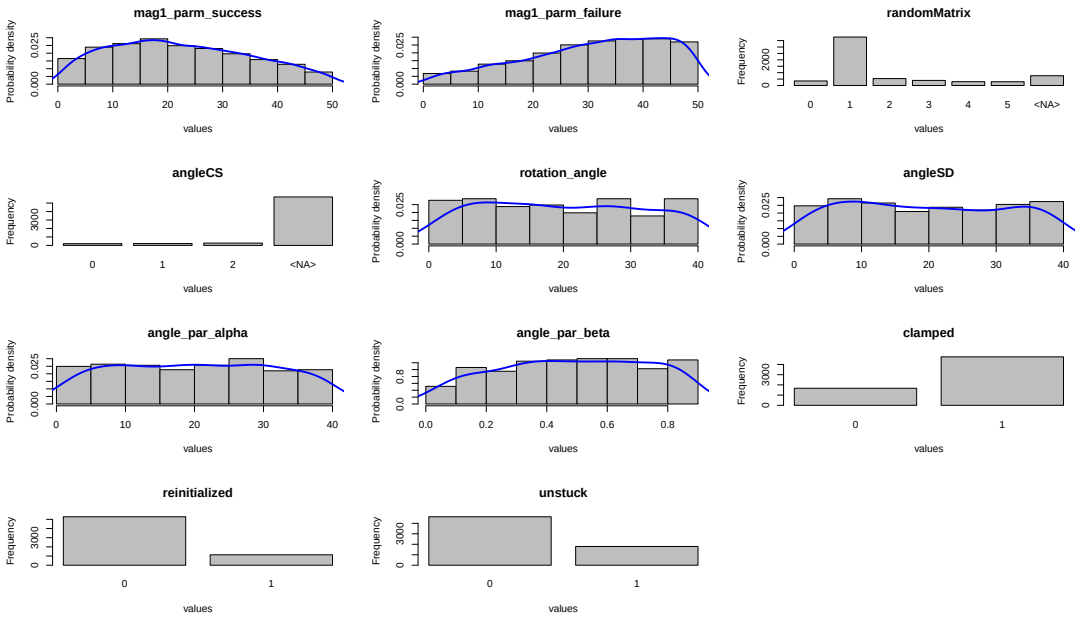


Figure B.3: Frequency of the sampled configurations of PSO-X_{hyb}

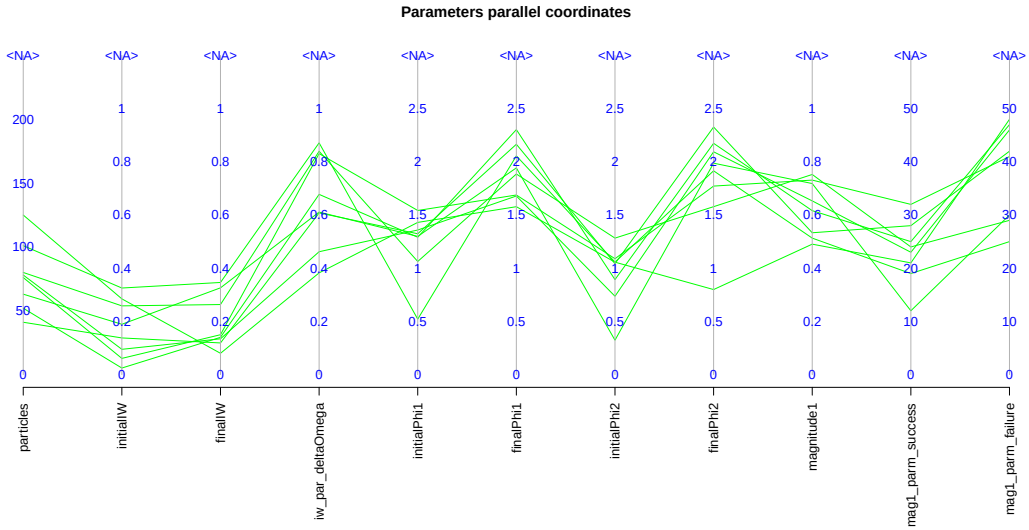
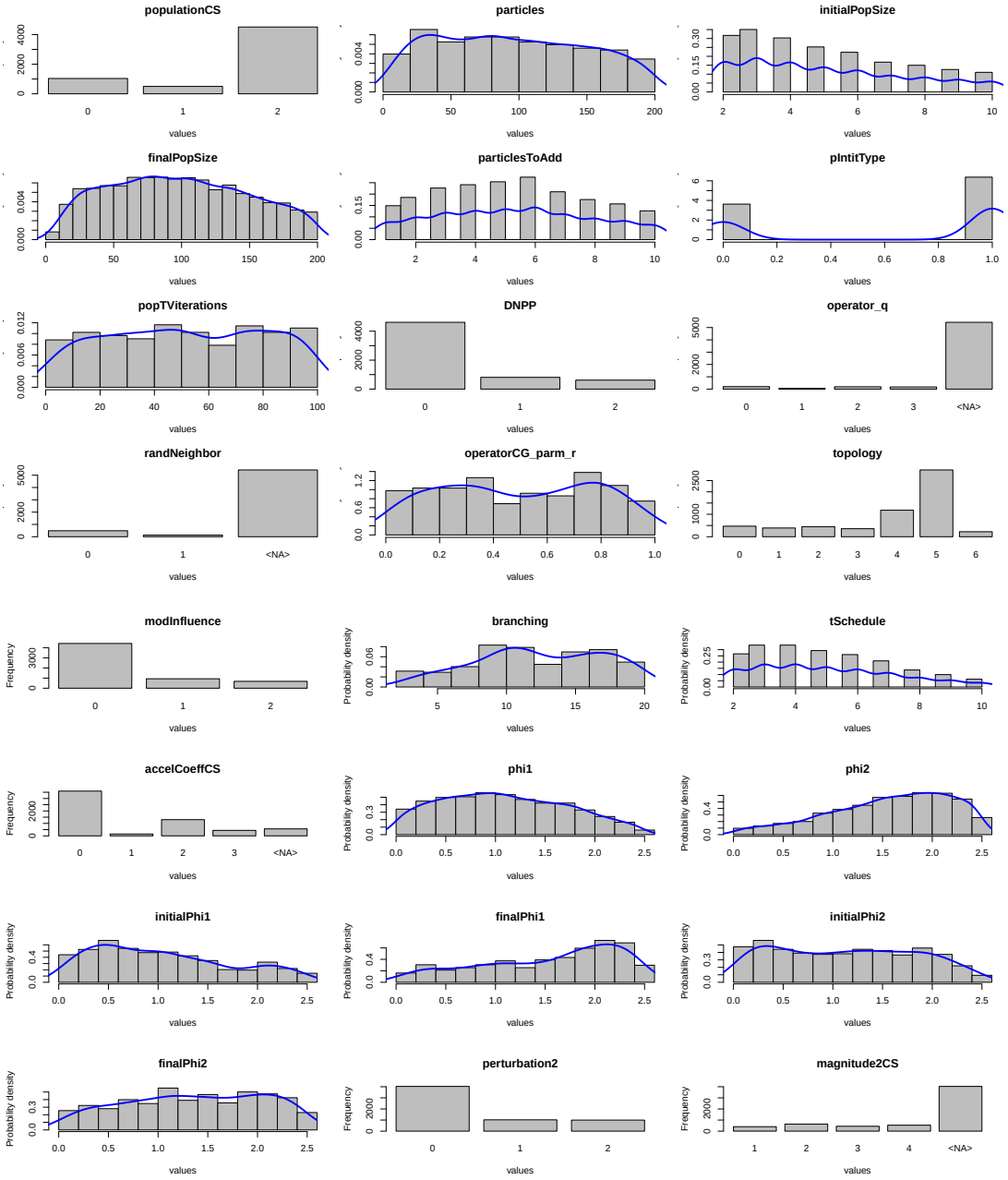
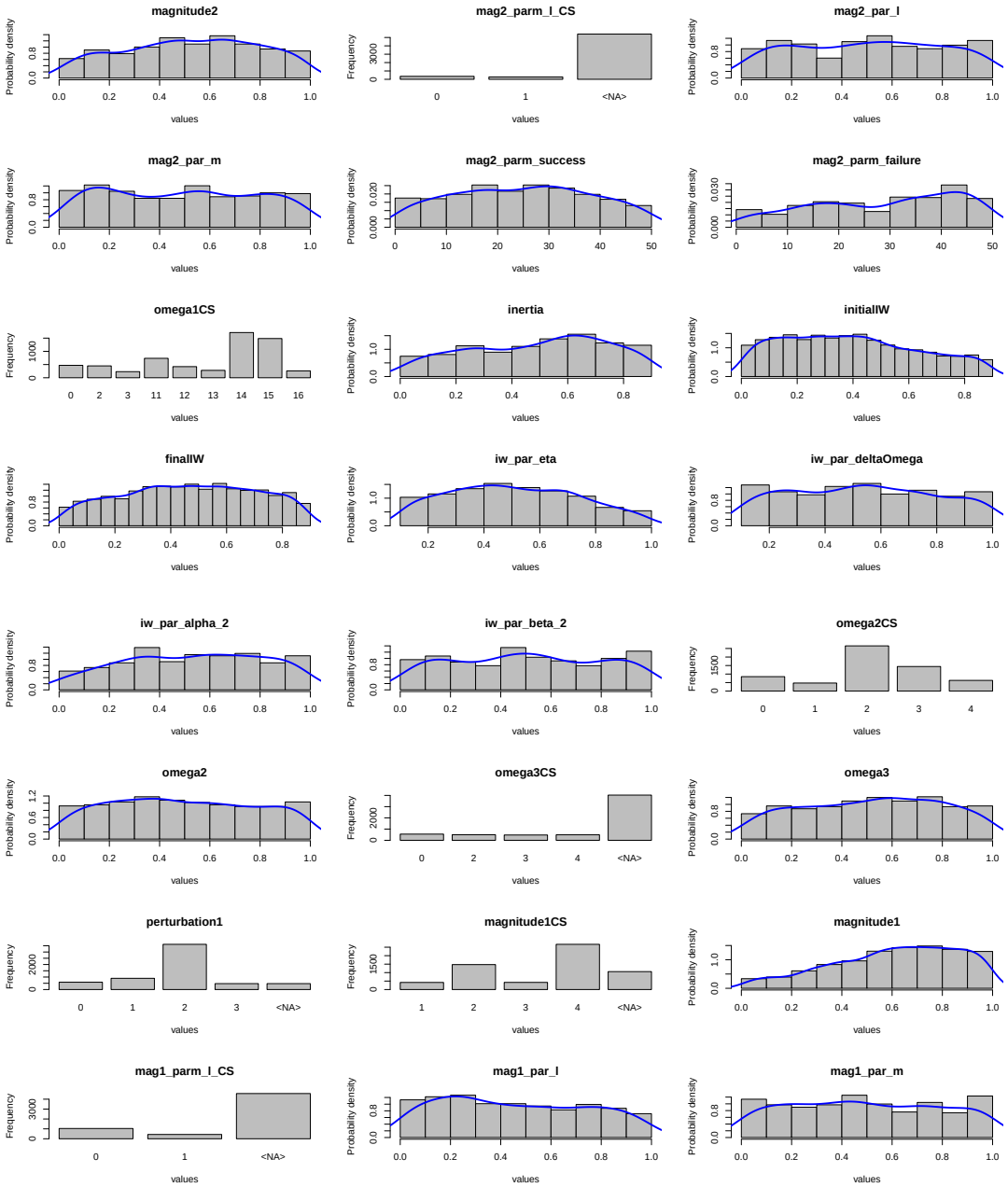


Figure B.4: Parameters interaction of PSO-X_{hyb}

PSO-X_{mul}


populationCS
particles
initialPopSize
finalPopSize
particlesToAdd
pIntitType
popTViterations
DNPP
operator_q
randNeighbor
operatorCG_parm_r
topology
modInfluence
branching
tSchedule
accelCoeffCS
phi1
phi2
initialPhi1
finalPhi1
initialPhi2
finalPhi2
perturbation2
magnitude2CS
values
Frequency
Probability density


magnitude2
mag2_parm_l_CS
mag2_par_l
mag2_par_m
mag2_parm_success
mag2_parm_failure
omega1CS
inertia
initialIW
finalIW
iw_par_eta
iw_par_deltaOmega
iw_par_alpha_2
iw_par_beta_2
omega2CS
omega2
omega3CS
omega3
perturbation1
magnitude1CS
magnitude1
mag1_parm_l_CS
mag1_par_l
mag1_par_m
Probability density
Frequency
values

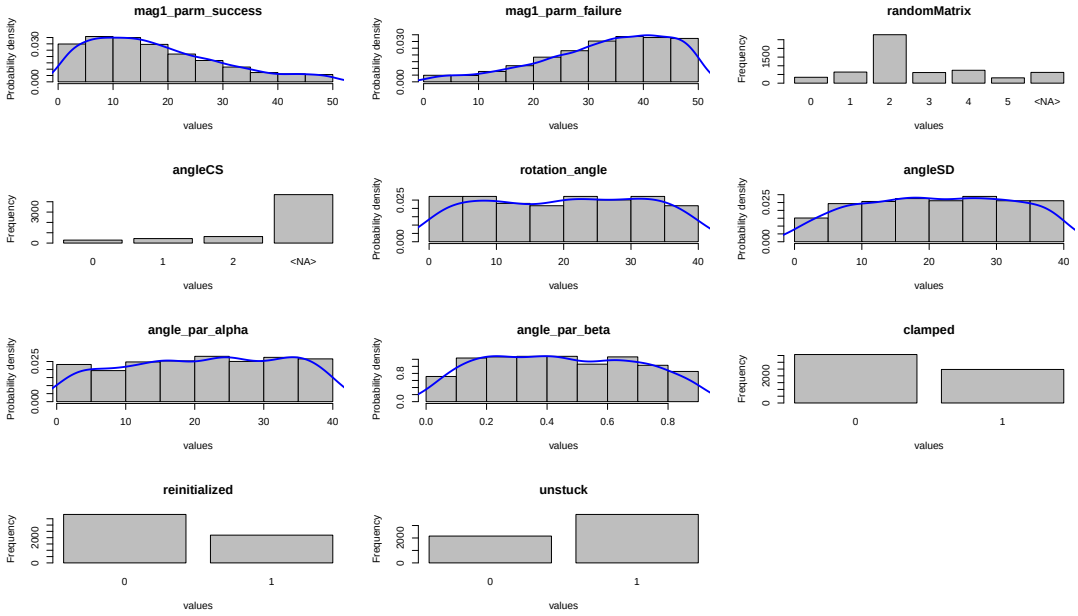


Figure B.5: Frequency of the sampled configurations of PSO-X_{mul}

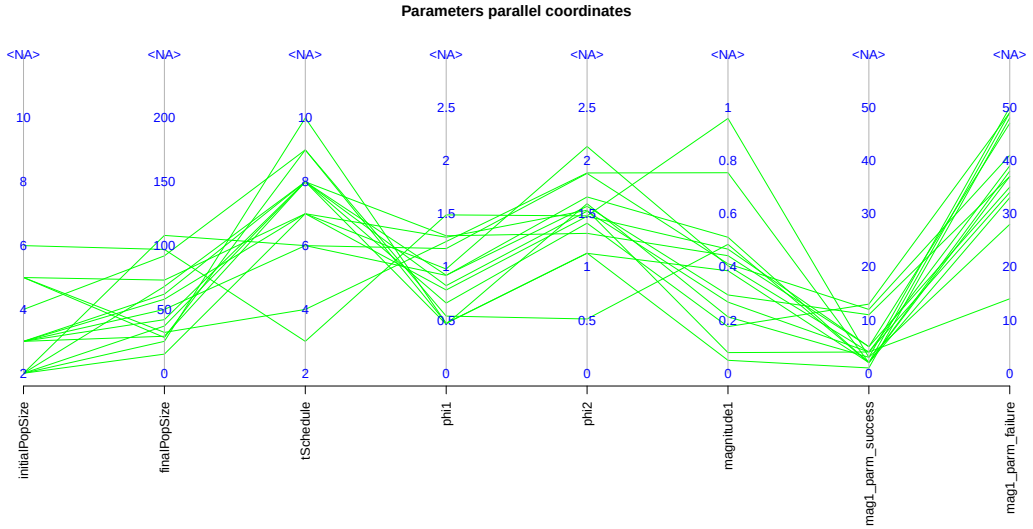
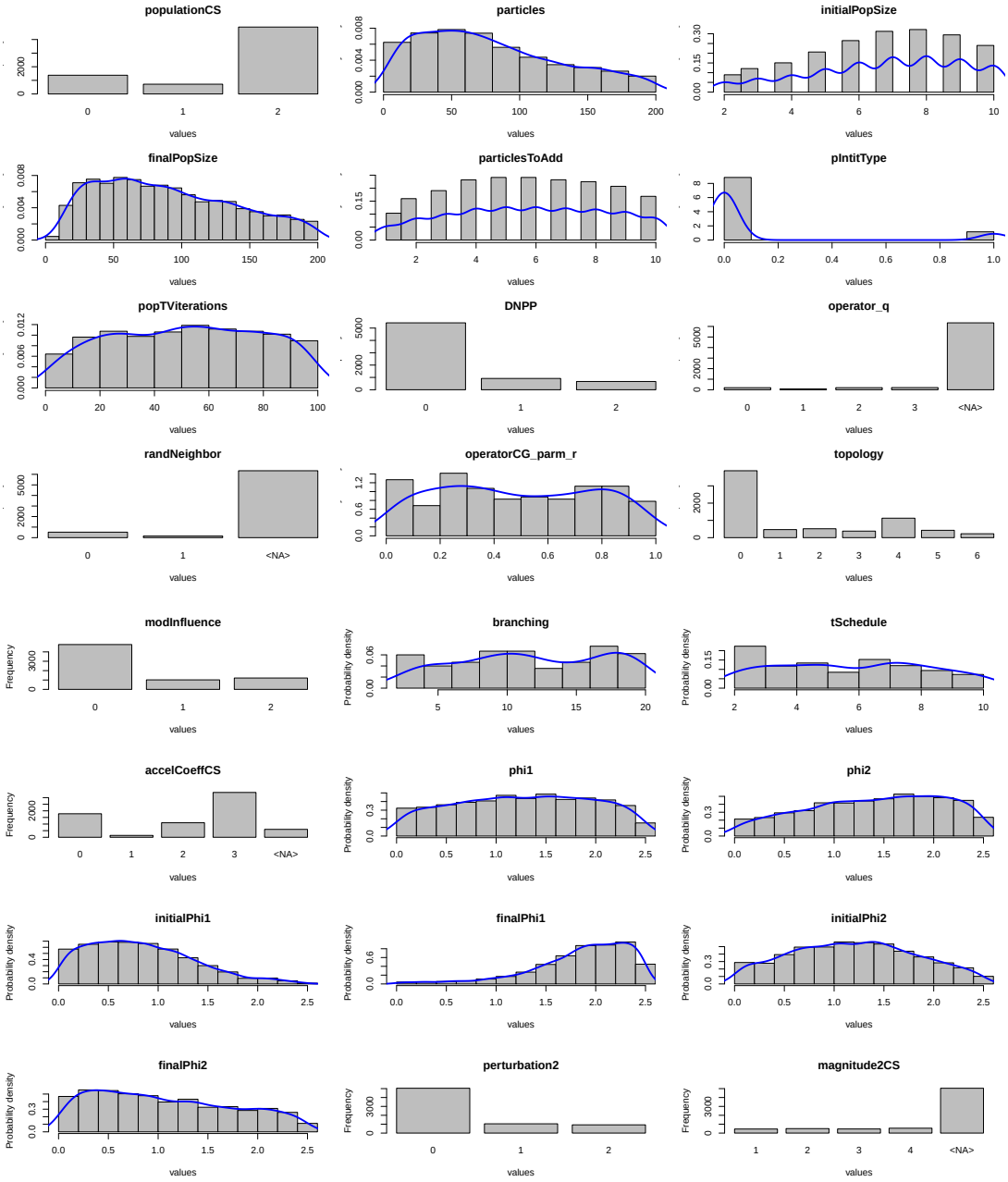
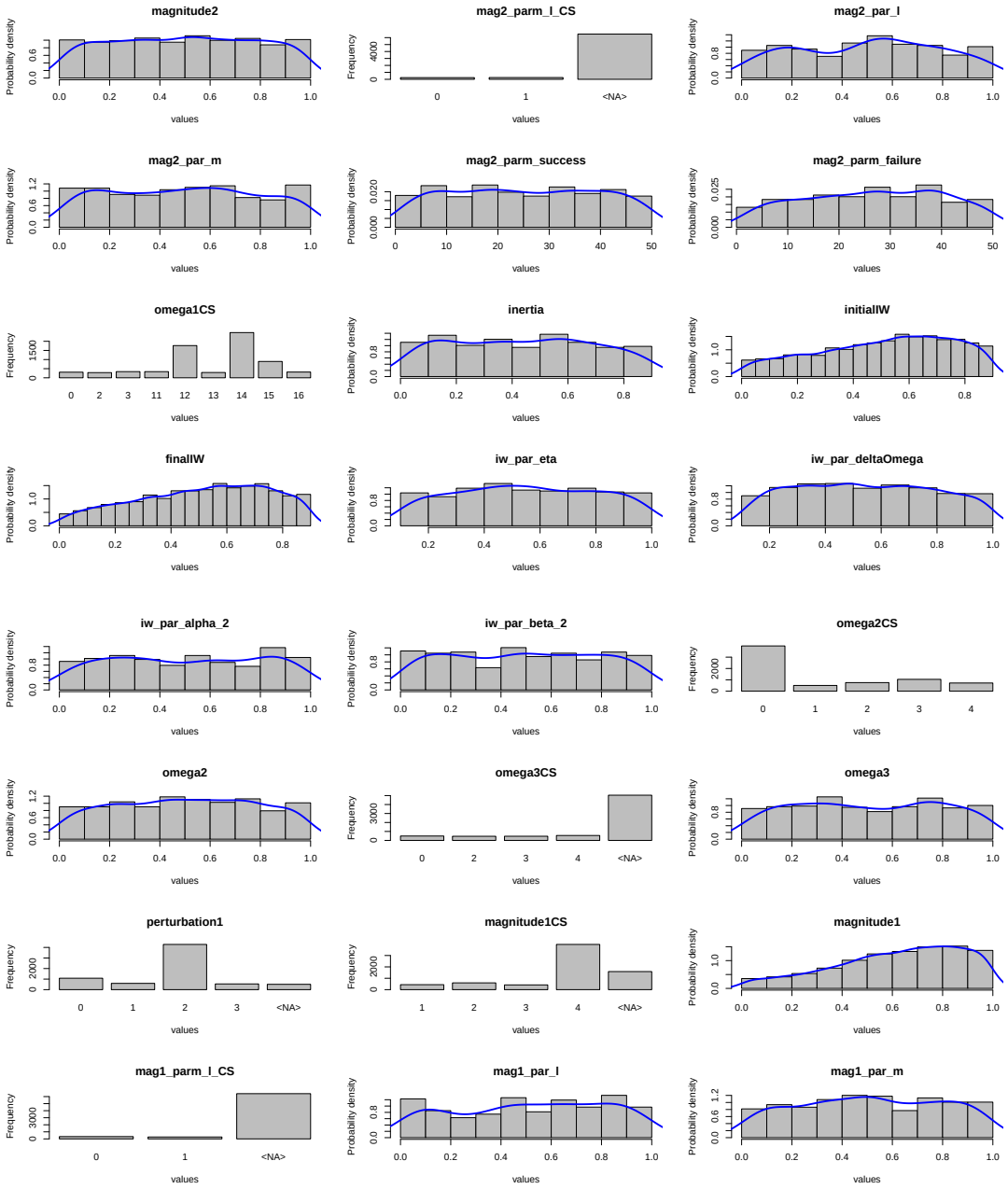


Figure B.6: Parameters interaction of PSO-X_{mul}

PSO-X_{uni}


populationCS
particles
initialPopSize
finalPopSize
particlesToAdd
pIntitType
popTViterations
DNPP
operator_q
randNeighbor
operatorCG_parm_r
topology
modInfluence
branching
tSchedule
accelCoeffCS
phi1
phi2
initialPhi1
finalPhi1
initialPhi2
finalPhi2
perturbation2
magnitude2CS
values
Frequency
Probability density


magnitude2
Probability density
values
mag2_parm_l_CS
Frequency
values
mag2_par_l
Probability density
values
mag2_par_m
Probability density
values
mag2_parm_success
Probability density
values
mag2_parm_failure
Probability density
values
omega1CS
Frequency
values
inertia
Probability density
values
initialIW
Probability density
values
finalIW
Probability density
values
iw_par_eta
Probability density
values
iw_par_deltaOmega
Probability density
values
iw_par_alpha_2
Probability density
values
iw_par_beta_2
Probability density
values
omega2CS
Frequency
values
omega2
Probability density
values
omega3CS
Frequency
values
omega3
Probability density
values
perturbation1
Frequency
values
magnitude1CS
Frequency
values
magnitude1
Probability density
values
mag1_parm_l_CS
Frequency
values
mag1_par_l
Probability density
values
mag1_par_m
Probability density
values

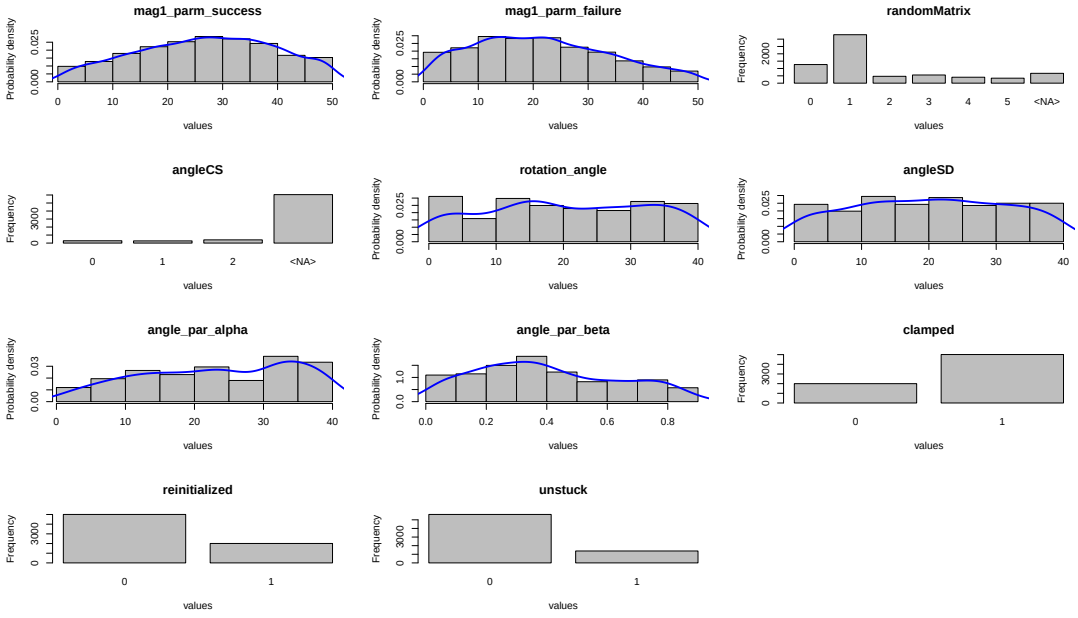


Figure B.7: Frequency of the sampled configurations of PSO-X_{uni}

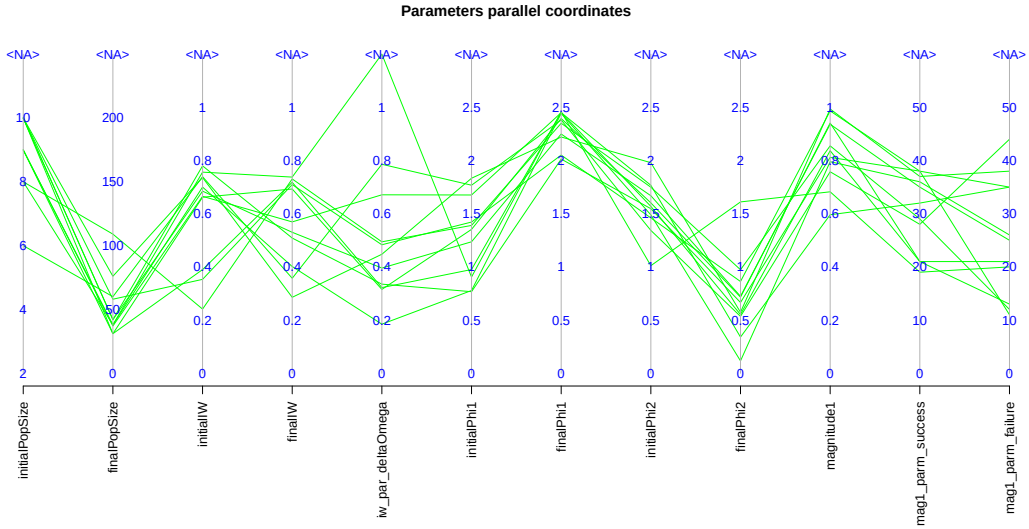
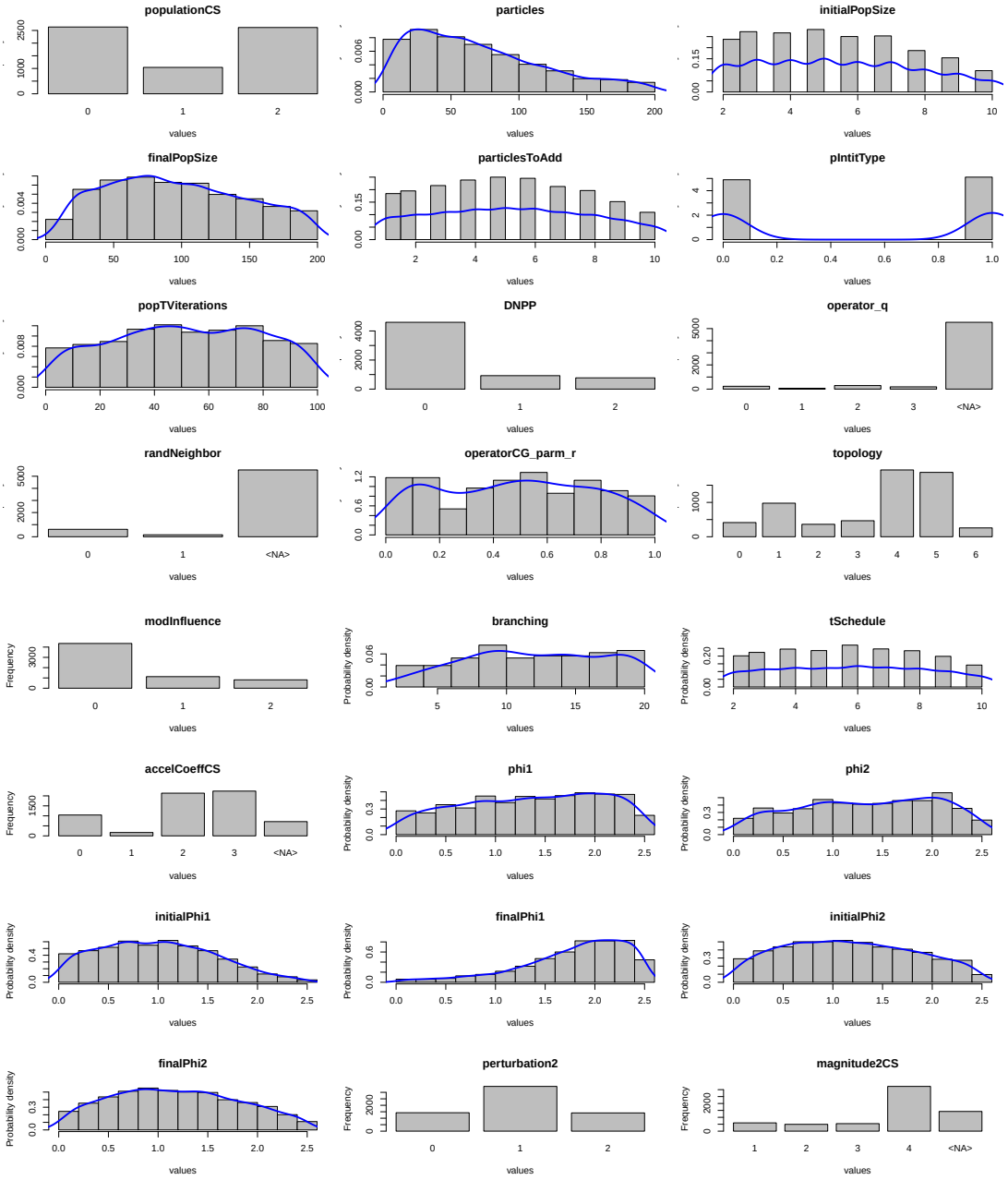
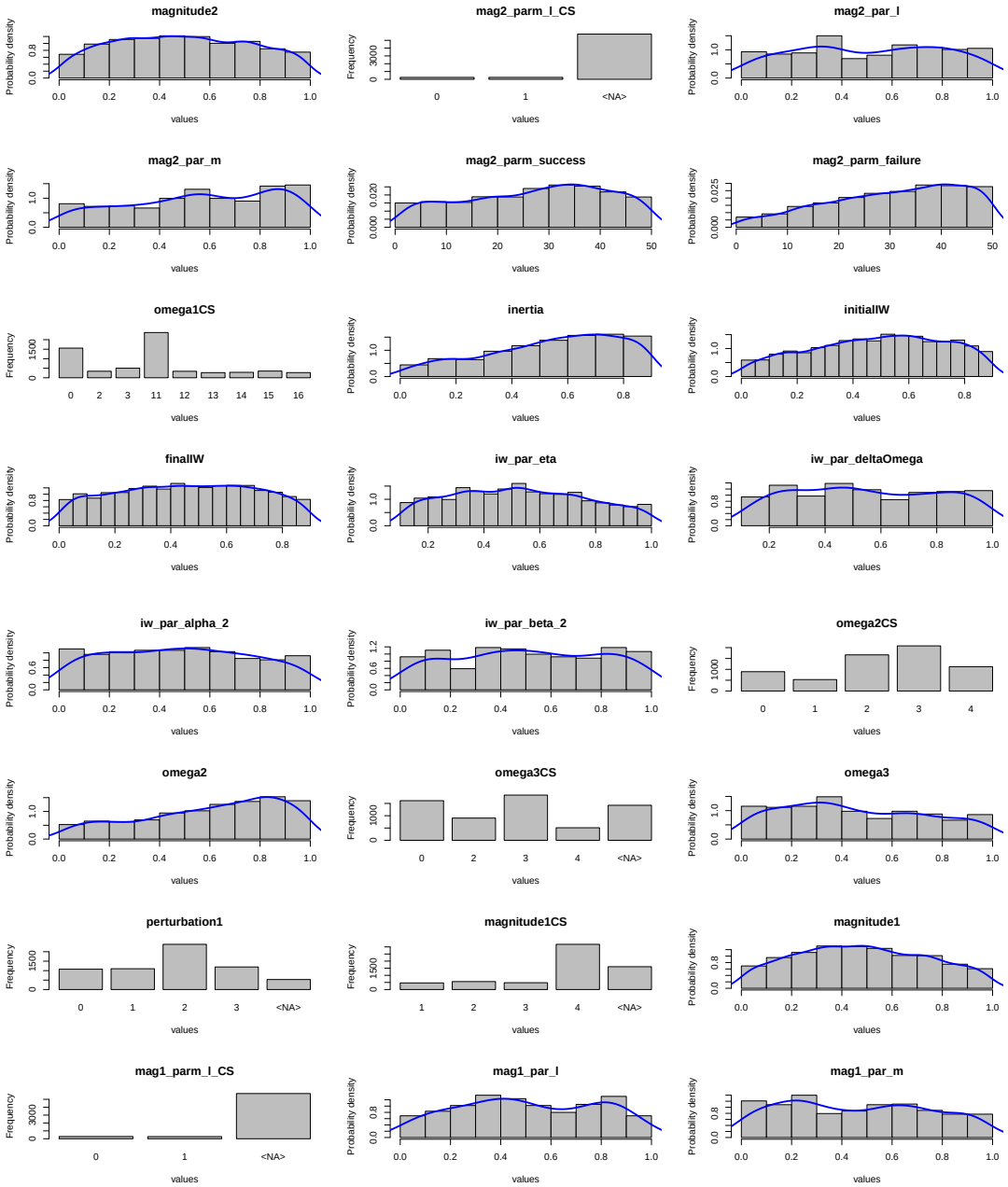


Figure B.8: Parameters interaction of PSO-X_{uni}

PSO-X_{cec}


populationCS
particles
initialPopSize
finalPopSize
particlesToAdd
pIntitType
popTViterations
DNPP
operator_q
randNeighbor
operatorCG_parm_r
topology
modInfluence
branching
tSchedule
accelCoeffCS
phi1
phi2
initialPhi1
finalPhi1
initialPhi2
finalPhi2
perturbation2
magnitude2CS
values
Frequency
Probability density


magnitude2
mag2_parm_l_CS
mag2_par_l
mag2_par_m
mag2_parm_success
mag2_parm_failure
omega1CS
inertia
initialIW
finalIW
iw_par_eta
iw_par_deltaOmega
iw_par_alpha_2
iw_par_beta_2
omega2CS
omega2
omega3CS
omega3
perturbation1
magnitude1CS
magnitude1
mag1_parm_l_CS
mag1_par_l
mag1_par_m
Probability density
Frequency
values
<NA>

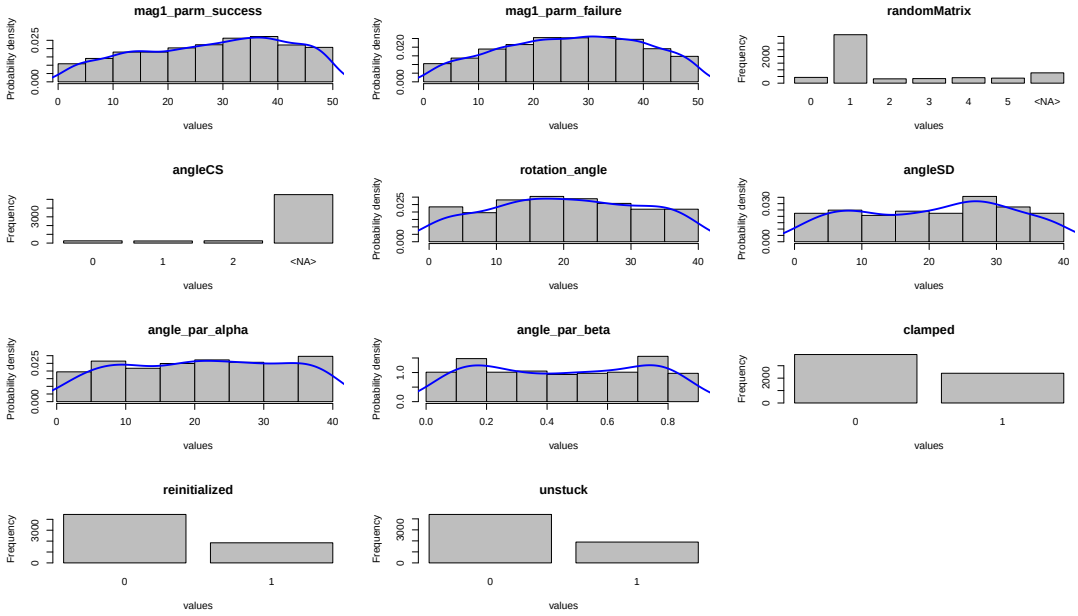


Figure B.9: Frequency of the sampled configurations of PSO-X_{cec}

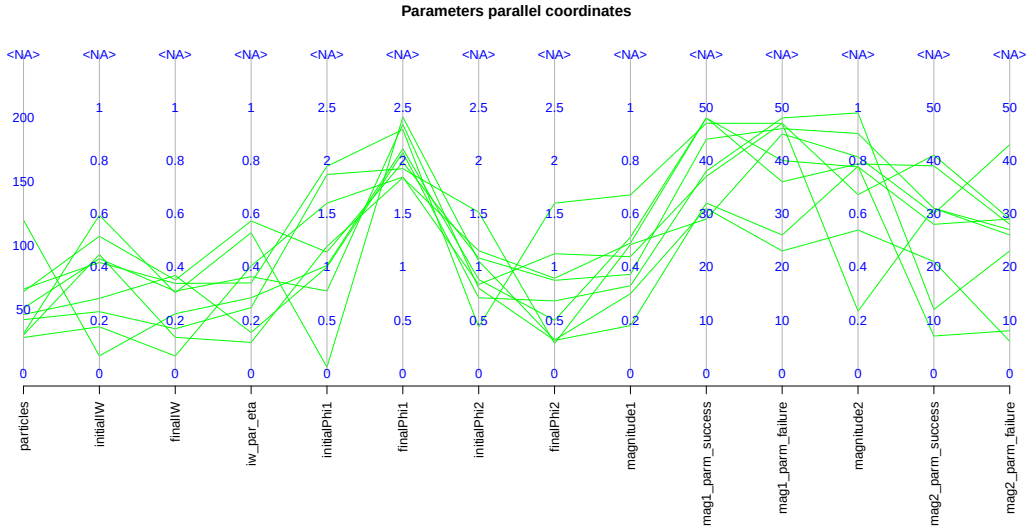
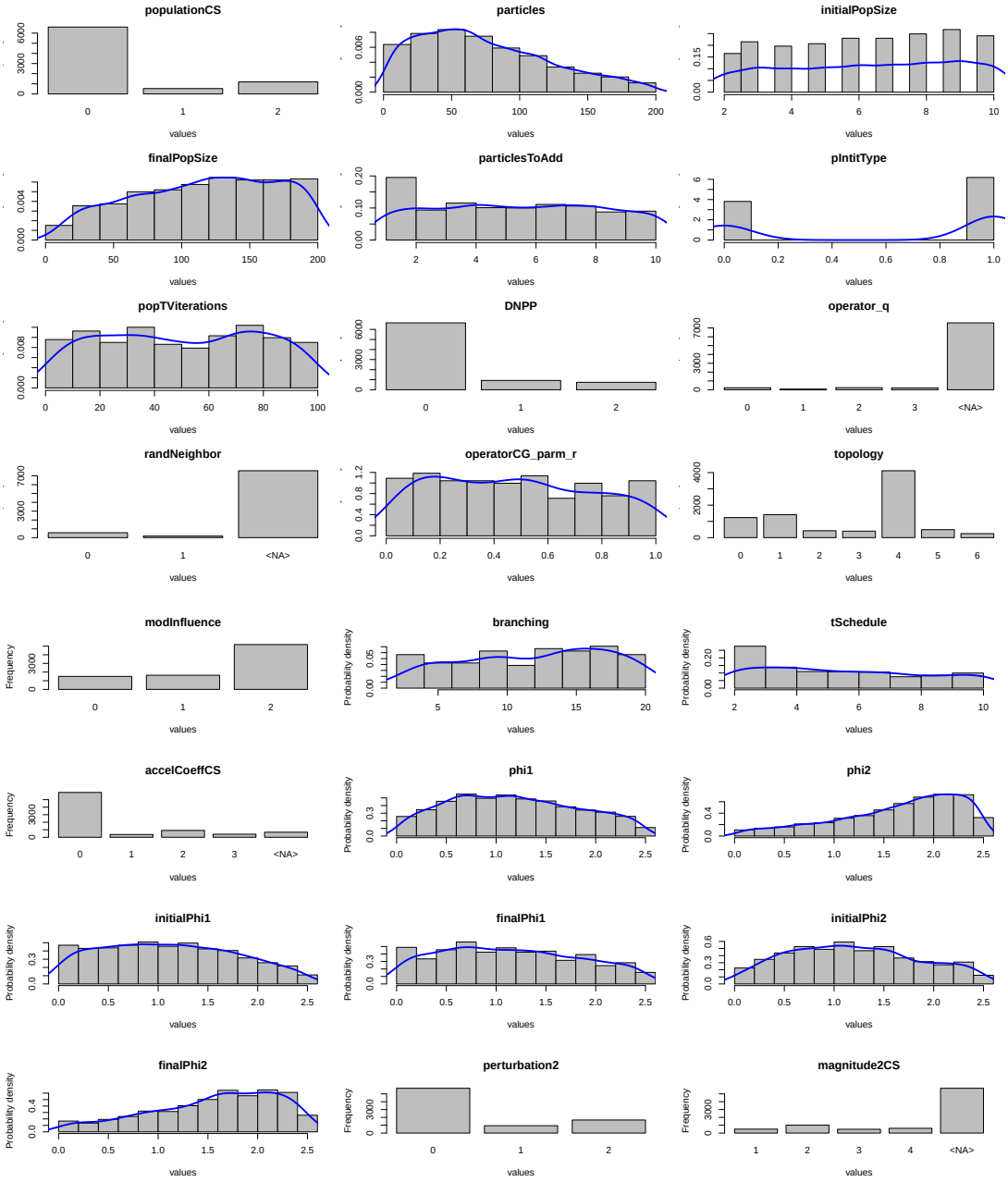
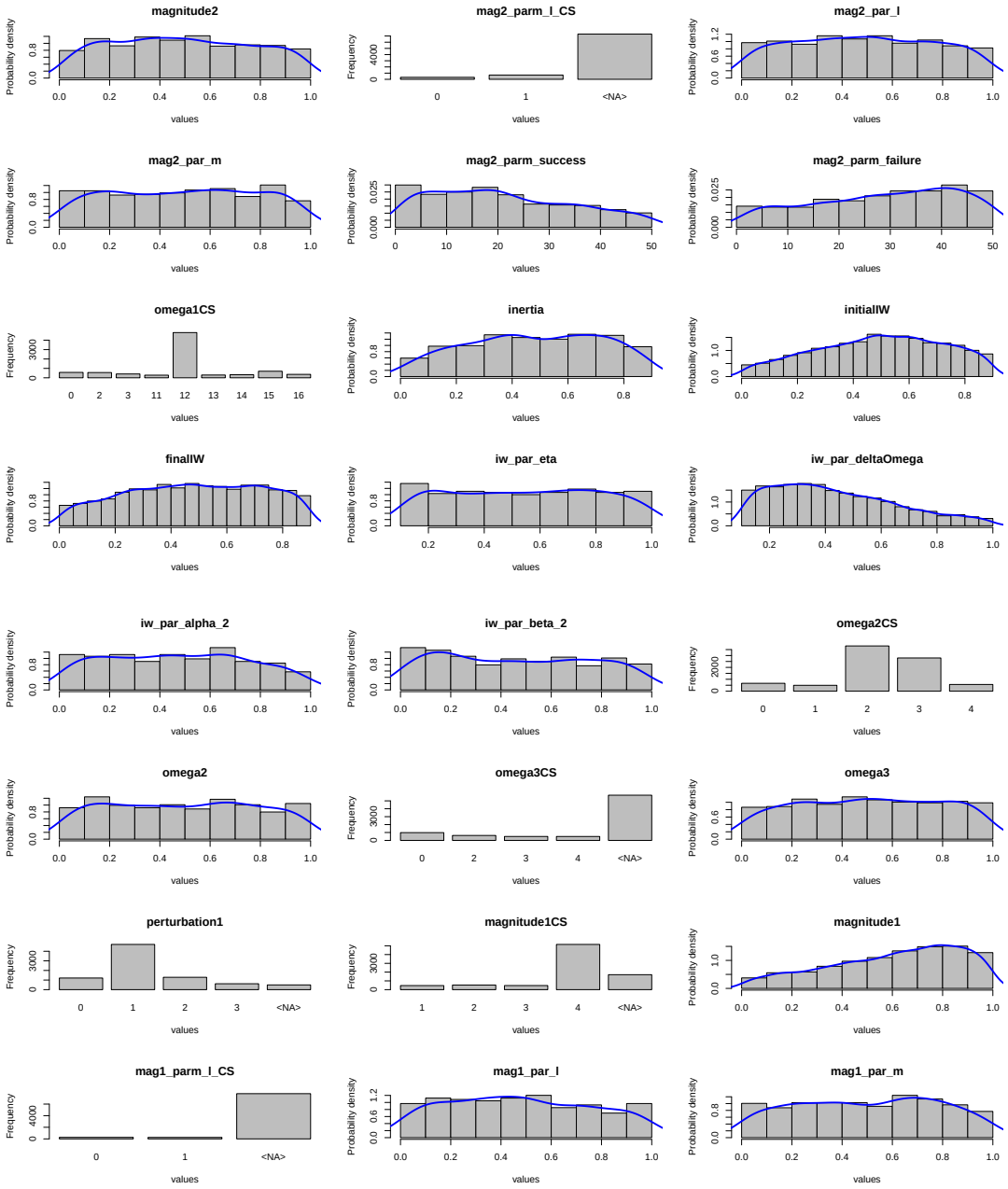


Figure B.10: Parameters interaction of PSO-X_{cec}

PSO-X_{soco}


populationCS
particles
initialPopSize
finalPopSize
particlesToAdd
pIntitType
popTViterations
DNPP
operator_q
randNeighbor
operatorCG_parm_r
topology
modInfluence
branching
tSchedule
accelCoeffCS
phi1
phi2
initialPhi1
finalPhi1
initialPhi2
finalPhi2
perturbation2
magnitude2CS
values
Frequency
Probability density


magnitude2
mag2_parm_l_CS
mag2_par_l
mag2_par_m
mag2_parm_success
mag2_parm_failure
omega1CS
inertia
initialW
finalW
iw_par_eta
iw_par_deltaOmega
iw_par_alpha_2
iw_par_beta_2
omega2CS
omega2
omega3CS
omega3
perturbation1
magnitude1CS
magnitude1
mag1_parm_l_CS
mag1_par_l
mag1_par_m
Probability density
Frequency
values

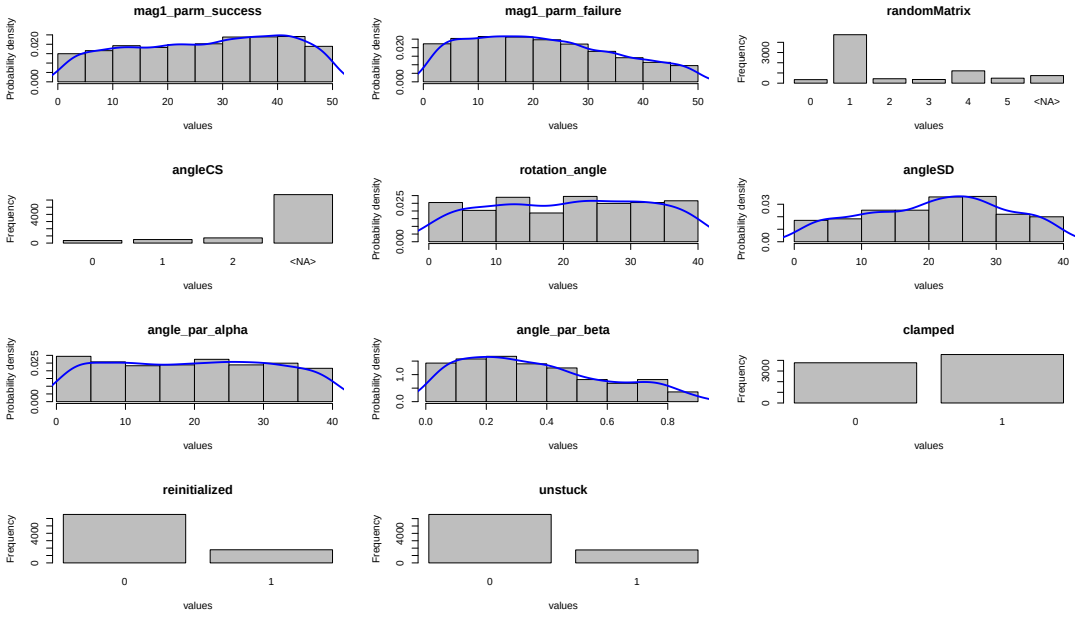


Figure B.11: Frequency of the sampled configurations of PSO-X_{soco}

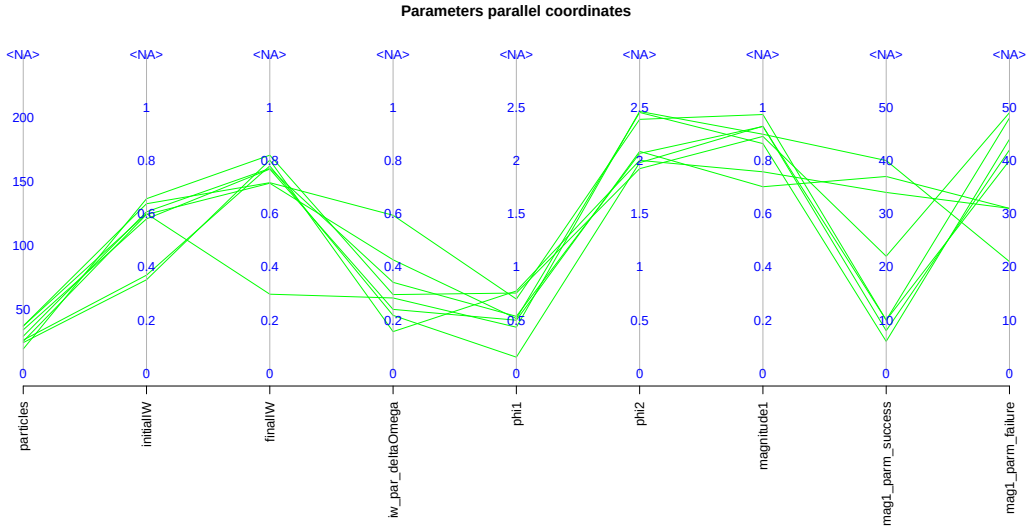


Figure B.12: Parameters interaction of PSO-X_{soco}

B.3 Algorithms of the Six PSO-*X* Implementation and of the Ten PSO Variants

In this section, we present in detail the algorithm of the six PSO-*X* implementations and of the ten PSO variants. We focus on showing how the formulation of the generalized velocity update rule (Equation 8.4) and the algorithm template for PSO (Algorithm 10) described in Chapter 8 can be used to instantiate these algorithms.

Algorithm 12 Standard PSO, StaPSO$_{tnd}$

Require: Pop-constant, Top-Von Neumann, MoI-best-of-neighborhood, DNPP-rectangular, Mtx-random diagonal, $\text{Pert}_{\text{info}} = \text{Pert}_{\text{rand}} = \text{none}$, $\text{pop} = 34$, $\omega_1 = 0.6615$, $\varphi_1 = 2.3706$, $\varphi_2 = 0.8914$.

1: INITIALIZE(Pop-constant, Top-Von Neumann, MoI-best-of-neighborhood)
2: $t \leftarrow 0$
3: **repeat**
4: **for** $i \leftarrow 1$ **to** size(`swarm`) **do**
5: $\vec{v}_{t+1}^{\,i} \leftarrow \omega_{1t}\,\vec{v}_t^{\,i} + \varphi_1 U_{1t}^i(\vec{p}_t^{\,i} - \vec{x}_t^{\,i}) + \varphi_2 U_{2t}^i(\vec{l}_t^{\,i} - \vec{x}_t^{\,i})$
6: $\vec{x}_{t+1}^{\,i} \leftarrow \vec{x}_t^{\,i} + \vec{v}_{t+1}^{\,i}$
7: **end for**
8: **for** $i \leftarrow 1$ **to** size(`swarm`) **do**
9: compute $f(\vec{x}_t^{\,i})$
10: update $\vec{p}_t^{\,i}$
11: **end for**
12: $\vec{g}_t \leftarrow$ best(`swarm`)
13: $t \leftarrow t + 1$
14: **until** $t = t_{max}$
15: **return** $\vec{g}_t$

Algorithm 13 PSO-X_{all}

Require: Pop-incremental, Init-random, Top-fully-connected, MoI-best-of-neighborhood, DNPP-rectangular, $\text{Pert}_{\text{info}}$-Lévy, PM-success rate, Mtx-random diagonal, velocity clamping, $\text{pop}_{ini} = 4$, $\text{pop}_{fin} = 20$, $\xi = 8$, $\omega_1 =$ convergence-based, $a = 0.7192$, $b = 0.9051$, $\omega_2 =$ random, $\omega_3 = 0$, $\varphi_1 = 1.7067$, $\varphi_2 = 2.2144$, PM $= 0.438$, $s_c = 11$ and $f_c = 40$

Initialize(Pop-incremental, Top-fully-connected, MoI-best-of-neighborhood)
$t \leftarrow 0$
repeat
  if (#successes $> s_c$) **then**
    PM $\leftarrow$ (PM $\cdot$ 2)
  end if
  if (#failures $> f_c$) **then**
    PM $\leftarrow$ (PM $\cdot$ 0.5)
  end if
  for $i \leftarrow 1$ **to** size(`swarm`) **do**
    $C_t^i \leftarrow \frac{|f(\vec{p}_{t-1}^{\,i}) - f(\vec{p}_t^{\,i})|}{f(\vec{p}_{t-1}^{\,i}) - f(\vec{p}_t^{\,i})}$
    $D_t^i \leftarrow \frac{|f(\vec{p}_t^{\,i}) - f(\vec{l}_t^{\,i})|}{f(\vec{p}_t^{\,i}) - f(\vec{l}_t^{\,i})}$
    $\omega_{1t}^i = 1 - \left| \frac{a - C_t^i}{(1 + D_t^i)(1+b)} \right|$
    $\omega_2 \leftarrow \mathcal{U}[0.5, 1]$
    $\vec{v}_{t+1}^{\,i} \leftarrow \omega_{1t}^i \vec{v}_t^{\,i} + \omega_2 \left(\varphi_1 U_{1t}^i \left(L_{\gamma_t}((\vec{p}_t^{\,i}), \text{PM}) - \vec{x}_t^{\,i} \right) + \varphi_2 U_{2t}^i \left(L_{\gamma_t}((\vec{l}_t^{\,i}), \text{PM}) - \vec{x}_t^{\,i} \right) \right)$
    apply velocity clamping
    $\vec{x}_{t+1}^{\,i} \leftarrow \vec{x}_t^{\,i} + \vec{v}_{t+1}^{\,i}$
  end for
  for $i \leftarrow 1$ **to** size(`swarm`) **do**
    compute $f(\vec{x}_t^{\,i})$
    update $\vec{p}_t^{\,i}$
  end for
  $\vec{g}_t \leftarrow$ best(`swarm`)
  if $(f(\vec{g}_t) < f(\vec{g}_{t-1}))$ **then**
    #successes $\leftarrow$ #successes $+ 1$
    #failures $\leftarrow 0$
  else
    #failures $\leftarrow$ #failures $+ 1$
    #successes $\leftarrow 0$
  end if
  `swarm` $\leftarrow$ UpdatePopulation(size(`swarm`), ξ, Pop-incremental)
  `swarm` $\leftarrow$ UpdateTopology(Top-fully-connected, MoI-best-of-neighborhood)
  $t \leftarrow t + 1$
until $t = t_{max}$
return $\vec{g}_t$

Algorithm 14 PSO-X_{hyb}

Require: Pop-constant, Top-Von Neumann, MoI-best-of-neighborhood, DNPP-rectangular, $\text{Pert}_{\text{info}}$-Lévy, PM-success rate, Mtx-random diagonal, velocity clamping, pop $= 41$, $\omega_1 =$ adaptive based onvelocity, $\omega_{1min} = 0.119$, $\omega_{1max} = 0.1378$, $\lambda = 0.608$, $\omega_2 = 1.0$, $\omega_3 = 0$, AC-random, $\varphi_{1min} = 1.0429$, $\varphi_{1max} = 2.1653$, $\varphi_{2min} = 1.0429$, $\varphi_{2max} = 2.3275$, PM $= 0.5333$, $s_c = 28$ and $f_c = 42$

Initialize(Pop-constant, Top-Von Neumann, MoI-best-of-neighborhood)
$t \leftarrow 0$
repeat
  if (#successes $> s_c$) **then**
    PM $\leftarrow$ (PM $\cdot$ 2)
  end if
  if (#failures $> f_c$) **then**
    PM $\leftarrow$ (PM $\cdot$ 0.5)
  end if
  $v_t^{\text{ideal}} \leftarrow \frac{ub-lb}{2}\left(\frac{1+\cos\left(\pi \frac{t}{0.95 \cdot t_{max}}\right)}{2}\right)$
  for $i \leftarrow 1$ **to** size(`swarm`) **do**
    $\overline{v_t} \leftarrow \frac{1}{\text{size}(\texttt{swarm}) \cdot d} \sum_{i=1}^{\text{size}(\texttt{swarm})} \sum_{j=1}^{d} |v_t^{i,j}|$
    if ($\overline{v_t} \geq v_{t+1}^{\text{ideal}}$) **then**
      $\omega_{1t}^i \leftarrow \max(\omega_{1t-1}^i - \lambda, \omega_{1min})$
    else
      $\omega_{1t}^i \leftarrow \min(\omega_{1t-1}^i + \lambda, \omega_{1max})$
    end if
    $\varphi_{1t} \leftarrow \mathcal{U}[\varphi_{1min}, \varphi_{1max}]$
    $\varphi_{2t} \leftarrow \mathcal{U}[\varphi_{2min}, \varphi_{2max}]$
    $\vec{v}_{t+1}^i \leftarrow \omega_{1t}^i \vec{v}_t^i + \varphi_{1t} U_{1t}^i \left(L_{\gamma_t}((\vec{p}_t^i), \text{PM}) - \vec{x}_t^i\right) + \varphi_{2t} U_{2t}^i \left(L_{\gamma_t}((\vec{l}_t^i), \text{PM}) - \vec{x}_t^i\right)$
    apply velocity clamping
    $\vec{x}_{t+1}^i \leftarrow \vec{x}_t^i + \vec{v}_{t+1}^i$
  end for
  for $i \leftarrow 1$ **to** size(`swarm`) **do**
    compute $f(\vec{x}_t^i)$
    update $\vec{p}_t^i$
  end for
  $\vec{g}_t \leftarrow$ best(`swarm`)
  if ($f(\vec{g}_t) < f(\vec{g}_{t-1})$) **then**
    #successes $\leftarrow$ #successes $+ 1$
    #failures $\leftarrow 0$
  else
    #failures $\leftarrow$ #failures $+ 1$
    #successes $\leftarrow 0$
  end if
  $t \leftarrow t + 1$
until $t = t_{max}$
return $\vec{g}_t$

Algorithm 15 PSO-X_{mul}

Require: Pop-incremental, Init-horizontal, Top-time-varying, MoI-best-of-neighborhood, DNPP-rectangular, $\text{Pert}_{\text{info}}$-Lévy, PM-success rate, Mtx-random linear, stagnation detection, $\kappa = 300$, $\text{pop}_{ini} = 3$, $\text{pop}_{fin} = 50$, $\xi = 2$, $\omega_1 =$ success-based, $\omega_{1min} = 0.4$, $\omega_{1max} = 0.9$, $\omega_2 = 1.0$, $\omega_3 = 0$, AC-constant, $\varphi_1 = 0.92$, $\varphi_2 = 1.6577$, PM $= 0.5114$, $s_c = 2$ and $f_c = 33$

Initialize(Pop-incremental, Top-time-varying, MoI-best-of-neighborhood)
$t \leftarrow 0$
repeat
  if (#successes $> s_c$) **then**
    PM $\leftarrow$ (PM $\cdot$ 2)
  end if
  if (#failures $> f_c$) **then**
    PM $\leftarrow$ (PM $\cdot$ 0.5)
  end if
  for $i \leftarrow 1$ **to** size(swarm) **do**
    $S_t \leftarrow 0$
    if $f(\vec{p}_t^{\,i}) < f(\vec{p}_{t-1}^{\,i})$ **then**
      $S_t \leftarrow S_t + 1$
    end if
    $\omega_{1t} \leftarrow \omega_{1min} + (\omega_{1max} - \omega_{1min}) \frac{S_t}{n}$
  end for
  for $i \leftarrow 1$ **to** size(swarm) **do**
    $\vec{v}_{t+1}^{\,i} \leftarrow \omega_{1t}\, \vec{v}_t^{\,i} + \varphi_1 r_{1t}^i \left(L_{\gamma_t}((\vec{p}_t^{\,i}), \text{PM}) - \vec{x}_t^{\,i} \right) + \varphi_2 r_{2t}^i \left(L_{\gamma_t}((\vec{l}_t^{\,i}), \text{PM}) - \vec{x}_t^{\,i} \right)$
    $\vec{x}_{t+1}^{\,i} \leftarrow \vec{x}_t^{\,i} + \vec{v}_{t+1}^{\,i}$
  end for
  for $i \leftarrow 1$ **to** size(swarm) **do**
    compute $f(\vec{x}_t^{\,i})$
    update $\vec{p}_t^{\,i}$
  end for
  apply stagnation detection
  $\vec{g}_t \leftarrow$ best(swarm)
  if $(f(\vec{g}_t) < f(\vec{g}_{t-1}))$ **then**
    #successes $\leftarrow$ #successes $+ 1$
    #failures $\leftarrow 0$
  else
    #failures $\leftarrow$ #failures $+ 1$
    #successes $\leftarrow 0$
  end if
  swarm $\leftarrow$ UpdatePopulation(size(swarm), ξ, Pop-incremental)
  swarm $\leftarrow$ UpdateTopology(κ, Top-time-varying, MoI-best-of-neighborhood)
  $t \leftarrow t + 1$
until $t = t_{max}$
return $\vec{g}_t$

Algorithm 16 PSO-X_{uni}

Require: Pop-incremental, Init-random, Top-fully-connected, MoI-best-of-neighborhood, DNPP-rectangular, $\text{Pert}_{\text{info}}$-Lévy, PM-success rate, Mtx-random diagonal, velocity clamping. $\text{pop}_{ini} = 10$, $\text{pop}_{fin} = 58$, $\xi = 3$, $\omega_1 =$ adaptive based onvelocity, $\omega_{1min} = 0.3531$, $\omega_{1max} = 0.7095$, $\lambda = 0.4832$, $\omega_2 = \omega_1$, AC-random, $\varphi_{1min} = 1.4217$, $\varphi_{1max} = 2.051$, $\varphi_{2min} = 0.8626$, $\varphi_{2max} = 1.4609$, PM $= 0.9865$, $s_c = 38$ and $f_c = 11$

1: INITIALIZE(Pop-incremental, Top-fully-connected, MoI-best-of-neighborhood)
2: $t \leftarrow 0$
3: **repeat**
4: **if** (#successes $> s_c$) **then**
5: PM $\leftarrow$ (PM $\cdot$ 2)
6: **end if**
7: **if** (#failures $> f_c$) **then**
8: PM $\leftarrow$ (PM $\cdot$ 0.5)
9: **end if**
10: $v_t^{\text{ideal}} \leftarrow \frac{ub-lb}{2}\left(\frac{1+\cos\left(\pi \frac{t}{0.95 \cdot t_{max}}\right)}{2}\right)$
11: **for** $i \leftarrow 1$ **to** size(swarm) **do**
12: $\overline{v_t} \leftarrow \frac{1}{\text{size(swarm)} \cdot d} \sum_{i=1}^{\text{size(swarm)}} \sum_{j=1}^{d} |v_t^{i,j}|$
13: **if** ($\overline{v_t} \geq v_{t+1}^{\text{ideal}}$) **then**
14: $\omega_{1t}^i \leftarrow \max(\omega_{1t-1}^i - \lambda, \omega_{1min})$
15: **else**
16: $\omega_{1t}^i \leftarrow \min(\omega_{1t-1}^i + \lambda, \omega_{1max})$
17: **end if**
18: $\omega_2 \leftarrow \omega_{1t}^i$
19: $\varphi_{1t} \leftarrow \mathcal{U}[\varphi_{1min}, \varphi_{1max}]$
20: $\varphi_{2t} \leftarrow \mathcal{U}[\varphi_{2min}, \varphi_{2max}]$
21: $\vec{v}_{t+1}^i \leftarrow \omega_{1t}^i \vec{v}_t^i + \omega_2 \left(\varphi_{1t} U_{1t}^i \left(L_{\gamma_t}((\vec{p}_t^i), \text{PM}) - \vec{x}_t^i\right) + \varphi_{2t} U_{2t}^i \left(L_{\gamma_t}((\vec{l}_t^i), \text{PM}) - \vec{x}_t^i\right)\right)$
22: apply velocity clamping
23: $\vec{x}_{t+1}^i \leftarrow \vec{x}_t^i + \vec{v}_{t+1}^i$
24: **end for**
25: **for** $i \leftarrow 1$ **to** size(swarm) **do**
26: compute $f(\vec{x}_t^i)$
27: update $\vec{p}_t^i$
28: **end for**
29: $\vec{g}_t \leftarrow$ best(swarm)
30: **if** ($f(\vec{g}_t) < f(\vec{g}_{t-1})$) **then**
31: #successes $\leftarrow$ #successes $+ 1$
32: #failures $\leftarrow 0$
33: **else**
34: #failures $\leftarrow$ #failures $+ 1$
35: #successes $\leftarrow 0$
36: **end if**
37: swarm $\leftarrow$ UPDATEPOPULATION(size(swarm), ξ, Pop-incremental)
38: swarm $\leftarrow$ UPDATETOPOLOGY(Top-fully-connected, MoI-best-of-neighborhood)
39: $t \leftarrow t + 1$
40: **until** $t = t_{max}$
41: **return** $\vec{g}_t$

Algorithm 17 PSO-X_{cec}

Require: Pop-constant, Top-Von Neumann, MoI-best-of-neighborhood, DNPP-rectangular, $\text{Pert}_{\text{info}}$-Lévy, PM-success rate, $\text{Pert}_{\text{rand}}$-noisy, PM-success rate, Mtx-random diagonal. pop $= 42$, $\omega_1 =$ self-regulating, $\omega_{1min} = 0.1673$, $\omega_{1max} = 0.2317$, $\eta = 0.2468$, $\omega_2 =$ random, $\omega_3 =$ random, AC-random, $\varphi_{1min} = 1.8684$, $\varphi_{1max} = 1.9233$, $\varphi_{2min} = 0.2802$, $\varphi_{2max} = 1.5143$, $\text{PM}_1 = 0.4837$, $s_{c1} = 29$, $f_{c1} = 45$, $\text{PM}_2 = 0.8139$, $s_{c2} = 30$ and $f_{c2} = 43$

1: INITIALIZE(Pop-constant, Top-Von Neumann, MoI-best-of-neighborhood)
2: $t \leftarrow 0$
3: **repeat**
4: **if** (#successes $> s_{c1}$) **then**
5: $\text{PM}_1 \leftarrow (\text{PM}_1 \cdot 2)$
6: **end if**
7: **if** (#failures $> f_{c1}$) **then**
8: $\text{PM}_1 \leftarrow (\text{PM}_1 \cdot 0.5)$
9: **end if**
10: **if** (#successes $> s_{c2}$) **then**
11: $\text{PM}_2 \leftarrow (\text{PM}_2 \cdot 2)$
12: **end if**
13: **if** (#failures $> f_{c2}$) **then**
14: $\text{PM}_2 \leftarrow (\text{PM}_2 \cdot 0.5)$
15: **end if**
16: **for** $i \leftarrow 1$ **to** size(swarm) **do**
17: **if** $(\vec{p}_t^{\,i} = \vec{g}_t)$ **then**
18: $\omega_{1t}^i \leftarrow \omega_{1t-1}^i \eta \frac{(\omega_{1max} - \omega_{1min})}{t_{max}})$
19: **else**
20: $\omega_{1t}^i \leftarrow \omega_{1t-1}^i - \frac{(\omega_{1max} - \omega_{1min})}{t_{max}})$
21: **end if**
22: $\omega_2 \leftarrow \mathcal{U}[0.5, 1]$, $\omega_3 \leftarrow \mathcal{U}[0.5, 1]$
23: $\varphi_{1t} \leftarrow \mathcal{U}[\varphi_{1min}, \varphi_{1max}]$
24: $\varphi_{2t} \leftarrow \mathcal{U}[\varphi_{2min}, \varphi_{2max}]$
25: $\vec{v}_{t+1}^{\,i} \leftarrow \omega_{1t}^i \vec{v}_t^{\,i} + \omega_2 \left(\varphi_{1t} U_{1t}^i \left(L_{\gamma_t}((\vec{p}_t^{\,i}), \text{PM}_1) - \vec{x}_t^{\,i} \right) + \varphi_{2t} U_{2t}^i \left(L_{\gamma_t}((\vec{l}_t^{\,i}), \text{PM}_1) - \vec{x}_t^{\,i} \right) \right) + \omega_3 \left(\mathcal{U}[-\text{PM}_2/2, \text{PM}_2/2] \right)$
26: $\vec{x}_{t+1}^{\,i} \leftarrow \vec{x}_t^{\,i} + \vec{v}_{t+1}^{\,i}$
27: **end for**
28: **for** $i \leftarrow 1$ **to** size(swarm) **do**
29: compute $f(\vec{x}_t^{\,i})$
30: update $\vec{p}_t^{\,i}$
31: **end for**
32: $\vec{g}_t \leftarrow$ best(swarm)
33: **if** $(f(\vec{g}_t) < f(\vec{g}_{t-1}))$ **then**
34: #successes $\leftarrow$ #successes $+ 1$
35: #failures $\leftarrow 0$
36: **else**
37: #failures $\leftarrow$ #failures $+ 1$
38: #successes $\leftarrow 0$
39: **end if**
40: $t \leftarrow t + 1$
41: **until** $t = t_{max}$
42: **return** $\vec{g}_t$

Algorithm 18 PSO-X_{soco}

Require: Pop-constant, Top-ring, MoI-ranked fully informed, DNPP-rectangular, $\text{Pert}_{\text{info}}$-Gaussian, PM-success rate, Mtx-random diagonal, velocity clamping. pop $= 19$, $\omega_1 =$ adaptive based onvelocity, $\omega_{1min} = 0.6564$, $\omega_{1max} = 0.8201$, $\lambda = 0.2959$, $\omega_2 = \omega_1$, $\omega_3 = 0$, AC-constant, $\varphi_1 = 0.7542$, $\varphi_2 = 1.9235$, PM $= 0.8907$, $s_c = 22$ and $f_c = 49$

1: INITIALIZE(Pop-constant, Top-ring, MoI-ranked fully informed)
2: $t \leftarrow 0$
3: **repeat**
4: **if** (#successes $> s_c$) **then**
5: PM $\leftarrow$ (PM $\cdot$ 2)
6: **end if**
7: **if** (#failures $> f_c$) **then**
8: PM $\leftarrow$ (PM $\cdot$ 0.5)
9: **end if**
10: $v_t^{\text{ideal}} \leftarrow \frac{ub-lb}{2}\left(\frac{1+\cos\left(\pi \frac{t}{0.95 \cdot t_{max}}\right)}{2}\right)$
11: **for** $i \leftarrow 1$ **to** size(swarm) **do**
12: $\overline{v_t} \leftarrow \frac{1}{\text{size(swarm)} \cdot d} \sum_{i=1}^{\text{size(swarm)}} \sum_{j=1}^{d} |v_t^{i,j}|$
13: **if** $(\overline{v_t} \geq v_{t+1}^{\text{ideal}})$ **then**
14: $\omega_{1t}^i \leftarrow \max(\omega_{1t-1}^i - \lambda, \omega_{1min})$
15: **else**
16: $\omega_{1t}^i \leftarrow \min(\omega_{1t-1}^i + \lambda, \omega_{1max})$
17: **end if**
18: $\omega_2 \leftarrow \omega_{1t}^i$, $\vec{v}_{t+1}^i \leftarrow (\omega_{1t}^i \, \vec{v}_t^i)$, $c_2 \leftarrow \varphi_2$ $R^i \leftarrow$ RANK(I^i),
19: **for** $k \leftarrow 1$ **to** $|R^i|$ **do**
20: **if** $k = 1$ **then**
21: $\vec{v}_{t+1}^i \leftarrow \vec{v}_{t+1}^i + \omega_2 \left(\varphi_1 U_{2t}^i \left(L_{\gamma_t}((\vec{p}_t^{\,i} \in R^{i,k}), \text{PM}) - \vec{x}_t^i\right)\right)$
22: **else**
23: $c_2 \leftarrow \frac{c_2}{2}$
24: $\vec{v}_{t+1}^i \leftarrow \vec{v}_{t+1}^i + \omega_2 \left(c_2 U_{1t}^i \left(L_{\gamma_t}((\vec{p}_t^{\,i} \in R^{i,k}), \text{PM}) - \vec{x}_t^i\right)\right)$
25: **end if**
26: **end for**
27: apply velocity clamping
28: $\vec{x}_{t+1}^i \leftarrow \vec{x}_t^i + \vec{v}_{t+1}^i$
29: **end for**
30: **for** $i \leftarrow 1$ **to** size(swarm) **do**
31: compute $f(\vec{x}_t^i)$
32: update $\vec{p}_t^{\,i}$
33: **end for**
34: $\vec{g}_t \leftarrow$ best(swarm)
35: **if** $(f(\vec{g}_t) < f(\vec{g}_{t-1}))$ **then**
36: #successes $\leftarrow$ #successes $+ 1$, #failures $\leftarrow 0$
37: **else**
38: #failures $\leftarrow$ #failures $+ 1$, #successes $\leftarrow 0$
39: **end if**
40: $t \leftarrow t + 1$
41: **until** $t = t_{max}$
42: **return** $\vec{g}_t$

Algorithm 19 Enhanced rotation invariant PSO, ERiPSO$_{dft}$

Require: Pop-constant, Top-fully-connected, MoI-best-of-neighborhood, DNPP-rectangular, Mtx-Euclidean rotation, α-adaptive, $\text{Pert}_{\text{info}} = \text{Pert}_{\text{rand}} = \text{none}$, $\text{pop} = 20$, $\omega_1 = 0.7213475$, $\omega_2 = 1.0$, $\omega_3 = 0$, AC-random, $\varphi_{1min} = 0$, $\varphi_{1max} = 2.05$, $\varphi_{2min} = 0$, $\varphi_{2max} = 2.05$, Mtx-Euclidean rotation_{all} with α-adaptive and $\zeta = 30$ and $\rho = 0.01$.

Initialize(Pop-constant, Top-fully-connected, MoI-best-of-neighborhood)
$t \leftarrow 0$
repeat
  for $i \leftarrow 1$ **to** size(swarm) **do**
    $ir_t \leftarrow 0$
    for $i \leftarrow 1$ **to** size(swarm) **do**
      if $(\vec{p}_t^{\,i} < \vec{p}_{t-1}^{\,i})$ **then**
        $ir_t \leftarrow ir_t + 1$
      end if
    end for
    $\varphi_{1t} \leftarrow \mathcal{U}[\varphi_{1min}, \varphi_{1max}]$
    $\varphi_{2t} \leftarrow \mathcal{U}[\varphi_{2min}, \varphi_{2max}]$
    $\sigma = \frac{\zeta \times ir_t}{\sqrt{d}} + \rho$
    $\vec{v}_{t+1}^{\,i} \leftarrow 0$
    for $k \leftarrow 1$ **to** $|I^i|$ **do**
      $\vec{u}^{i,k} \leftarrow (\vec{p}_t^{\,k} - \vec{x}_t^{\,i})$
      for $m \leftarrow 1$ **to** $(d-1)$ **do**
        for $n \leftarrow m+1$ **to** d **do**
          compute $[r_{mn}]_k$ with angle σ
          $\vec{u}^{i,k} \leftarrow [r_{mn}]_k \, \vec{u}^{i,k}$
        end for
      end for
      if $(\vec{p}_t^{\,k} = \vec{l}_t^{\,i})$ **then**
        $\vec{v}_{t+1}^{\,i} \leftarrow \vec{v}_{t+1}^{\,i} + \varphi_{1t} \vec{u}_k^{\,i}$
      else
        $\vec{v}_{t+1}^{\,i} \leftarrow \vec{v}_{t+1}^{\,i} + \varphi_{2t} \vec{u}_k^{\,i}$
      end if
    end for
    $\vec{v}_{t+1}^{\,i} \leftarrow \vec{v}_{t+1}^{\,i} + \omega_{1t}^i \, \vec{v}_t^{\,i}$
    apply velocity clamping
    $\vec{x}_{t+1}^{\,i} \leftarrow \vec{x}_t^{\,i} + \vec{v}_{t+1}^{\,i}$
  end for
  for $i \leftarrow 1$ **to** size(swarm) **do**
    compute $f(\vec{x}_t^{\,i})$
    update $\vec{p}_t^{\,i}$
  end for
  $\vec{g}_t \leftarrow$ best(swarm)
  $t \leftarrow t + 1$
until $t = t_{max}$
return $\vec{g}_t$

Algorithm 20 Fully informed PSO, FiPSO$_{tnd}$

Require: Pop-constant, Top-ring, MoI-fully informed, DNPP-rectangular, Mtx-random diagonal, $\text{Pert}_{\text{info}} = \text{Pert}_{\text{rand}} = \text{none}$, pop $= 20$, $\omega_1 = \omega_2 = 0.729843788$, $\omega_3 = 0$, $\varphi_1 = 2.1864$ and $\varphi_2 = 2.3156$

1: INITIALIZE(Pop-constant, Top-ring, MoI-fully informed)
2: $t \leftarrow 0$
3: $\varphi \leftarrow \varphi_1 + \varphi_2$
4: **repeat**
5: **for** $i \leftarrow 1$ **to** size(swarm) **do**
6: $\varphi_k \leftarrow \frac{\varphi}{|I_t^i|}$
7: $\vec{v}_{t+1}^{\,i} = \omega_{1t}\vec{v}_t^{\,i} + \omega_{2t}\sum_{k\in I_t^i}^{n}\left(\varphi_k U_t^{i,k}(\vec{p}_t^{\,i} - \vec{x}_t^{\,i})\right)$
8: $\vec{x}_{t+1}^{\,i} \leftarrow \vec{x}_t^{\,i} + \vec{v}_{t+1}^{\,i}$
9: **end for**
10: **for** $i \leftarrow 1$ **to** size(swarm) **do**
11: compute $f(\vec{x}_t^{\,i})$
12: update $\vec{p}_t^{\,i}$
13: **end for**
14: $\vec{g}_t \leftarrow$ best(swarm)
15: $t \leftarrow t + 1$
16: **until** $t = t_{max}$
17: **return** $\vec{g}_t$

Algorithm 21 Frankenstein's PSO, FraPSO$_{dft}$

Require: Pop-constant, Top-time-varying, MoI-fully informed, DNPP-rectangular, Mtx-random diagonal, $\text{Pert}_{\text{info}} = \text{Pert}_{\text{rand}} = \text{none}$, $\kappa = 60$, pop $= 60$, $\omega_1 =$ linear decreasing, $\omega_{1min} = 0.4$, $\omega_{1max} = 0.9$, $t_{schd} = 10$, $\omega_2 = 0.7298$, $\omega_3 = 0$, $\varphi_1 = 2.0$ and $\varphi_2 = 2.0$

1: INITIALIZE(Pop-constant, Top-time-varying, MoI-fully informed)
2: **if** $t_{schd} \neq 0$ **then**
3: $t_{schd} \leftarrow t_{schd} \cdot \text{pop}$
4: **else**
5: $t_{schd} \leftarrow t_{max}$
6: **end if**
7: $t \leftarrow 0$
8: $\varphi \leftarrow \varphi_1 + \varphi_2$
9: **repeat**
10: **if** $(t \leq t_{schd})$ **then**
11: $\omega_{1t} \leftarrow \omega_2 \left(\frac{t_{schd} - t}{t_{schd}} (\omega_{1max} - \omega_{1min}) + \omega_{1min} \right)$
12: **else**
13: $\omega_{1t} \leftarrow \omega_2 \cdot \omega_{1min}$
14: **end if**
15: **for** $i \leftarrow 1$ **to** size(`swarm`) **do**
16: $\varphi_k \leftarrow \frac{\varphi}{|I_t^i|}$
17: $\vec{v}_{t+1}^{\,i} \leftarrow \omega_{1t} \vec{v}_t^{\,i} + \omega_2 \sum_{k \in I_t^i}^{n} \left(\varphi_k U_t^{i,k} (\vec{p}_t^{\,i} - \vec{x}_t^{\,i}) \right)$
18: $\vec{x}_{t+1}^{\,i} \leftarrow \vec{x}_t^{\,i} + \vec{v}_{t+1}^{\,i}$
19: **end for**
20: **for** $i \leftarrow 1$ **to** size(`swarm`) **do**
21: compute $f(\vec{x}_t^{\,i})$
22: update $\vec{p}_t^{\,i}$
23: **end for**
24: $\vec{g}_t \leftarrow$ best(`swarm`)
25: `swarm` $\leftarrow$ UPDATETOPOLOGY(κ, Top-time-varying, MoI-fully informed)
26: $t \leftarrow t + 1$
27: **until** $t = t_{max}$
28: **return** $\vec{g}_t$

Algorithm 22 Gaussian "bare-bones" PSO, GauPSO$_{tnd}$

Require: Pop-constant, Top-time-varying, MoI-random informant, DNPP-Gaussian, $\text{Pert}_{\text{info}} = \text{Pert}_{\text{rand}} = \text{none}$, $\kappa = 150$ pop $= 30$, $\omega_1 = 0$, $\omega_2 = 1.0$ and $\omega_3 = 0$

1: INITIALIZE(Pop-constant, Top-time-varying, MoI-random informant)
2: $t \leftarrow 0$
3: **repeat**
4: **for** $i \leftarrow 1$ **to** size(swarm) **do**
5: select random k from I^i
6: $\vec{v}^i_{t+1} \leftarrow \omega_{1t}\vec{v}^i_t + \mathcal{N}(\frac{\vec{p}^i_t + \vec{p}^k_t}{2}, |\vec{p}^i_t - \vec{p}^k_t|) - \vec{x}^i_t$
7: $\vec{x}^i_{t+1} \leftarrow \vec{x}^i_t + \vec{v}^i_{t+1}$
8: **end for**
9: **for** $i \leftarrow 1$ **to** size(swarm) **do**
10: compute $f(\vec{x}^i_t)$
11: update $\vec{p}^i_t$
12: **end for**
13: $\vec{g}_t \leftarrow$ best(swarm)
14: swarm $\leftarrow$ UPDATETOPOLOGY(κ, Top-time-varying, MoI-fully informed)
15: $t \leftarrow t + 1$
16: **until** $t = t_{max}$
17: **return** $\vec{g}_t$

Algorithm 23 Hierarchical PSO, HiePSO$_{tnd}$

Require: Pop-constant, Top-hierarchical, MoI-best-of-neighborhood, DNPP-rectangular, Mtx-random diagonal, $\text{Pert}_{\text{info}} = \text{Pert}_{\text{rand}} = \text{none}$, $bd = 2$, pop $= 114$, $\omega_1 =$ linear increasing, $\omega_{1min} = 0.3284$, $\omega_{1max} = 0.8791$, $\omega_2 = 1.0$, $\omega_3 = 0$, $\varphi_1 = 2.1105$ and $\varphi_2 = 1.0349$

1: INITIALIZE(Pop-constant, Top-hierarchical, MoI-best-of-neighborhood)
2: $t \leftarrow 0$
3: **repeat**
4: $\omega_{1t} \leftarrow \omega_{1min} - (\omega_{1min} - \omega_{1max})\frac{t}{t_{max}}$
5: **for** $i \leftarrow 1$ **to** size(swarm) **do**
6: $\vec{v}^i_{t+1} \leftarrow \omega_{1t}\,\vec{v}^i_t + \varphi_1 U^i_{1t}(\vec{p}^i_t - \vec{x}^i_t) + \varphi_2 U^i_{2t}(\vec{l}^i_t - \vec{x}^i_t)$
7: $\vec{x}^i_{t+1} \leftarrow \vec{x}^i_t + \vec{v}^i_{t+1}$
8: **end for**
9: **for** $i \leftarrow 1$ **to** size(swarm) **do**
10: compute $f(\vec{x}^i_t)$
11: update $\vec{p}^i_t$
12: **end for**
13: $\vec{g}_t \leftarrow$ best(swarm)
14: swarm $\leftarrow$ UPDATETOPOLOGY(bd, Top-hierarchical, MoI-best-of-neighborhood)
15: $t \leftarrow t + 1$
16: **until** $t = t_{max}$
17: **return** $\vec{g}_t$

Algorithm 24 Incremental PSO, IncPSO$_{tnd}$

Require: Pop-incremental, Init-horizontalTop-time-varying, Mol-best-of-neighborhood, DNPP-rectangular, Mtx-random diagonal, $\text{Pert}_{\text{info}} = \text{Pert}_{\text{rand}} =$ none, $\kappa = 2360$, $\text{pop}_{ini} = 5$, $\text{pop}_{fin} = 295$, $\xi = 10$, $\omega_1 = \omega_2 = 0.729843788$, $\omega_3 = 0$, $\varphi_1 = 1.9226$ and $\varphi_2 = 1.0582$

1: INITIALIZE(Pop-incremental, Top-time-varying, Mol-best-of-neighborhood)
2: $t \leftarrow 0$
3: **repeat**
4: **for** $i \leftarrow 1$ **to** size(`swarm`) **do**
5: $\vec{v}_{t+1}^{\,i} \leftarrow \omega_{1t}\,\vec{v}_t^{\,i} + \omega_2\Big(\varphi_1 U_{1t}^i(\vec{p}_t^{\,i} - \vec{x}_t^{\,i}) + \varphi_2 U_{2t}^i(\vec{l}_t^{\,i} - \vec{x}_t^{\,i})\Big)$
6: $\vec{x}_{t+1}^{\,i} \leftarrow \vec{x}_t^{\,i} + \vec{v}_{t+1}^{\,i}$
7: **end for**
8: **for** $i \leftarrow 1$ **to** size(`swarm`) **do**
9: compute $f(\vec{x}_t^{\,i})$
10: update $\vec{p}_t^{\,i}$
11: **end for**
12: $\vec{g}_t \leftarrow$ best(`swarm`)
13: `swarm` $\leftarrow$ UPDATEPOPULATION(size(`swarm`), ξ, Pop-incremental)
14: `swarm` $\leftarrow$ UPDATETOPOLOGY(κ, Top-hierarchical, Mol-best-of-neighborhood)
15: $t \leftarrow t + 1$
16: **until** $t = t_{max}$
17: **return** $\vec{g}_t$

Algorithm 25 Locally convergent rotation invariant PSO, LcRPSO$_{dft}$

Require: Pop-constant, Top-fully-connected, MoI-best-of-neighborhood, DNPP-rectangular, Mtx-random linear, Pert$_{\text{info}}$-Gaussian, PM-Euclidean distance, Pert$_{\text{rand}}$ = none, pop $= d$, $\omega_1 = 0.7298$, AC-random, $\varphi_{1min} = 0$, $\varphi_{1max} = 1.4962$, $\varphi_{2min} = 0$, $\varphi_{2max} = 1.4962$ and $\epsilon = 0.46461/d^{0.79}$.

1: INITIALIZE(Pop-constant, Top-fully-connected, MoI-best-of-neighborhood)
2: $t \leftarrow 0$
3: **repeat**
4: **for** $i \leftarrow 1$ **to** size(`swarm`) **do**
5: $\text{PM}_{t-1}^{i,1} \leftarrow 1.0$
6: $\text{PM}_{t-1}^{i,2} \leftarrow 1.0$
7: **end for**
8: **for** $i \leftarrow 1$ **to** size(`swarm`) **do**
9: **if** $(\vec{x}_t^i = \vec{p}_t^i)$ **then**
10: $\text{PM}_t^{i,1} \leftarrow \epsilon \cdot \text{PM}_{t-1}^{i,1}$
11: **else**
12: $\text{PM}_t^{i,1} \leftarrow \epsilon \cdot \sqrt{\sum_{j=1}^{d} (\vec{x}_t^{i,j} - \vec{p}_t^{i,j})^2}$
13: **end if**
14: **if** $(\vec{x}_t^i = \vec{l}_t^i)$ **then**
15: $\text{PM}_t^{i,2} \leftarrow \epsilon \cdot \text{PM}_{t-1}^{i,2}$
16: **else**
17: $\text{PM}_t^{i,2} \leftarrow \epsilon \cdot \sqrt{\sum_{j=1}^{d} (\vec{x}_t^{i,j} - \vec{l}_t^{i,j})^2}$
18: **end if**
19: $\varphi_{1t} \leftarrow \mathcal{U}[\varphi_{1min}, \varphi_{1max}]$
20: $\varphi_{2t} \leftarrow \mathcal{U}[\varphi_{2min}, \varphi_{2max}]$
21: $\vec{v}_{t+1}^i \leftarrow \omega_{1t}\, \vec{v}_t^i + \varphi_{1t} r_{1t}^i \left(\mathcal{N}(\vec{p}_t^i, \text{PM}_t^{i,1}) - \vec{x}_t^i \right) + \varphi_{2t} r_{2t}^i \left(\mathcal{N}(\vec{l}_t^i, \text{PM}_t^{i,2}) - \vec{x}_t^i \right)$
22: $\vec{x}_{t+1}^i \leftarrow \vec{x}_t^i + \vec{v}_{t+1}^i$
23: **end for**
24: **for** $i \leftarrow 1$ **to** size(`swarm`) **do**
25: compute $f(\vec{x}_t^i)$
26: update $\vec{p}_t^i$
27: **end for**
28: $\vec{g}_t \leftarrow$ best(`swarm`)
29: $t \leftarrow t + 1$
30: **until** $t = t_{max}$
31: **return** $\vec{g}_t$

Algorithm 26 Restart PSO, ResPSO$_{tnd}$

Require: Pop-constant, Top-ring, MoI-fully informed, DNPP-rectangular, Mtx-random diagonal, $\text{Pert}_{\text{info}} = \text{Pert}_{\text{rand}} = \text{none}$, pop $= 10$, $\omega_1 =$ linear decreasing, $\omega_{1min} = 0.2062$, $\omega_{1max} = 0.6446$, $\omega_2 = 1.0$, $\varphi_1 = 1.5014$, $\varphi_2 = 2.2955$

Initialize(Pop-constant, Top-ring, MoI-fully informed)
$t \leftarrow 0$
repeat
  for $i \leftarrow 1$ **to** size(swarm) **do**
    $\omega_{1t} \leftarrow \omega_{1min} - (\omega_{1min} - \omega_{1max}) \frac{t}{t_{max}}$
    $\vec{v}_{t+1}^{\,i} \leftarrow \omega_{1t}\, \vec{v}_t^{\,i} + \varphi_1 U_{1t}^i (\vec{p}_t^{\,i} - \vec{x}_t^{\,i}) + \varphi_2 U_{2t}^i (\vec{l}_t^{\,i} - \vec{x}_t^{\,i})$
    apply velocity clamping
    $\vec{x}_{t+1}^{\,i} \leftarrow \vec{x}_t^{\,i} + \vec{v}_{t+1}^{\,i}$
  end for
  for $i \leftarrow 1$ **to** size(swarm) **do**
    compute $f(\vec{x}_t^{\,i})$
    update $\vec{p}_t^{\,i}$
  end for
  apply particles reinitialization
  $\vec{g}_t \leftarrow$ best(swarm)
  $t \leftarrow t + 1$
until $t = t_{max}$
return $\vec{g}_t$

Algorithm 27 Standard PSO 2011, SPSO11$_{tnd}$

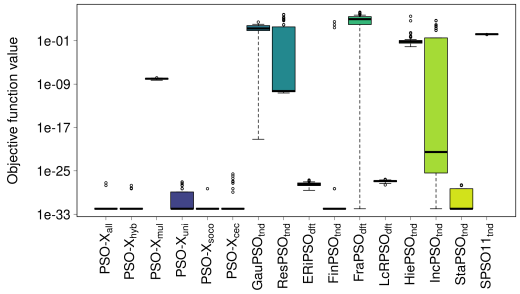
Require: Pop-constant, Top-time-varying, MoI-best-of-neighborhood, DNPP-spherical, Mtx-random diagonal, $\text{Pert}_{\text{info}} = \text{Pert}_{\text{rand}} = \text{none}$, $\kappa = 1085$, pop $= 155$, $\omega_1 = 0.6482$, $\omega_2 = 1.0$, $\varphi_1 = 2.2776$, $\varphi_2 = 2.1222$

Initialize(Pop-constant, Top-time-varying, MoI-best-of-neighborhood)
$t \leftarrow 0$
repeat
  for $i \leftarrow 1$ **to** size(swarm) **do**
    $\vec{P}_t^{\,i} \leftarrow \vec{x}_t^{\,i} + \varphi_{1t} U_t^k (\vec{p}_t^{\,i} - \vec{x}_t^{\,i})$
    $\vec{L}_t^{\,i} \leftarrow \vec{x}_t^{\,i} + \varphi_{2t} U_t^k (\vec{l}_t^{\,i} - \vec{x}_t^{\,i})$
    $\vec{c}_t^{\,i} \leftarrow \frac{\vec{x}_t^{\,i} + \vec{L}_t^{\,i} + \vec{P}_t^{\,i}}{3}$
    $\vec{v}_{t+1}^{\,i} \leftarrow \omega_{1t}\, \vec{v}_t^{\,i} + \mathcal{H}_i(\vec{c}_t^{\,i}, |\vec{c}_t^{\,i} - \vec{x}_t^{\,i}|) - \vec{x}_t^{\,i}$
    $\vec{x}_{t+1}^{\,i} \leftarrow \vec{x}_t^{\,i} + \vec{v}_{t+1}^{\,i}$
  end for
  for $i \leftarrow 1$ **to** size(swarm) **do**
    compute $f(\vec{x}_t^{\,i})$
    update $\vec{p}_t^{\,i}$
  end for
  $\vec{g}_t \leftarrow$ best(swarm)
  swarm $\leftarrow$ UpdateTopology(κ, Top-time-varying, MoI-best-of-neighborhood)
  $t \leftarrow t + 1$
until $t = t_{max}$
return $\vec{g}_t$

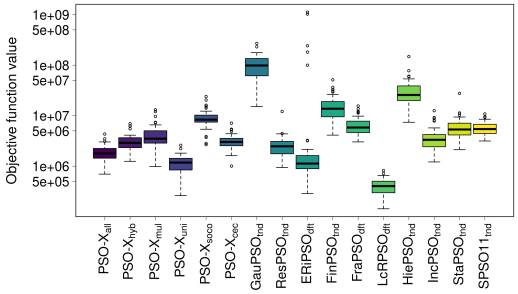
B.4 Distribution of the Results Obtained by the 16 Compared PSO Algorithms.

In this section, we report the results of the 16 PSO algorithms using box-plots. Each box-plot shows the distribution of the results obtained by executing each algorithm 50 times on each function. We present a total 90 box-plots, the first 50 correspond to the results obtained on the version of the functions with $d = 50$, and the remaining 40 to the results obtained on the version with $d = 100$.

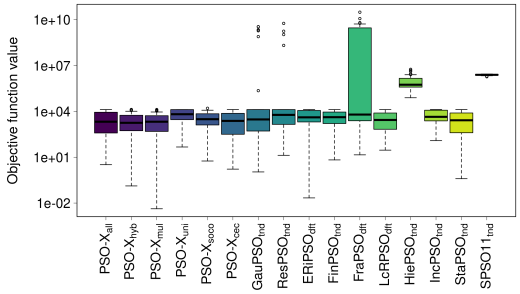
50 dimensions



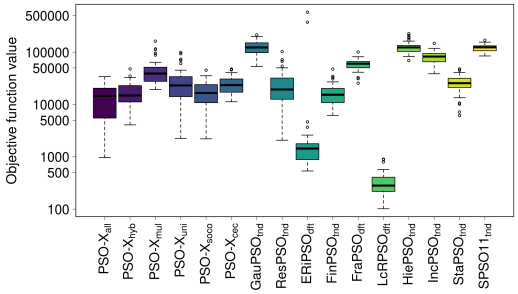
(f_1) Shifted Sphere - SOCO'10



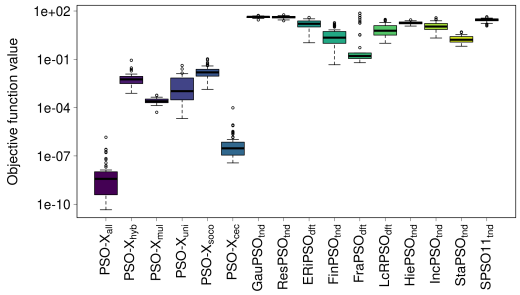
(f_2) Shifted Rotated High Conditioned Elliptic-CEC'14



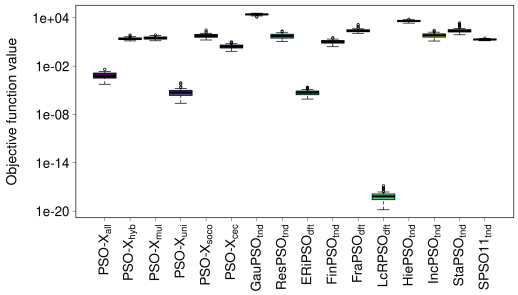
(f_3) Shifted Rotated Bent Cigar - CEC'14



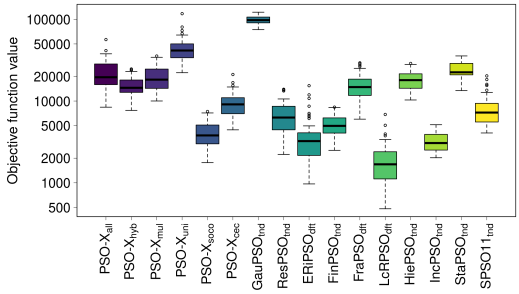
(f_4) Shifted Rotated Discus - CEC'14



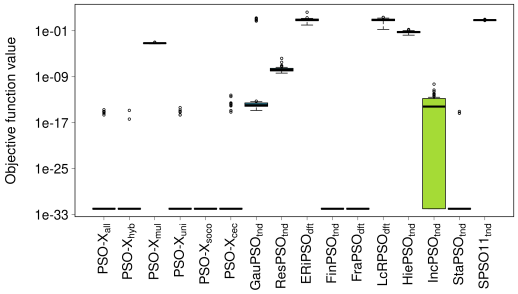
(f_5) Shifted Schwefel 22.1 - SOCO'10



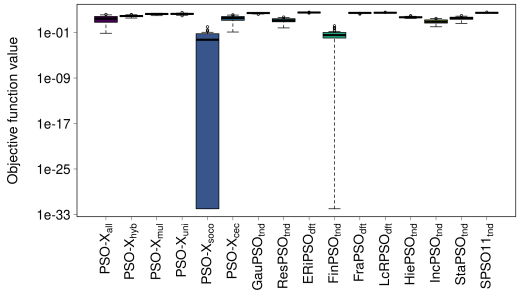
(f_6) Rotated Schwefel 1.2 - SOCO'10



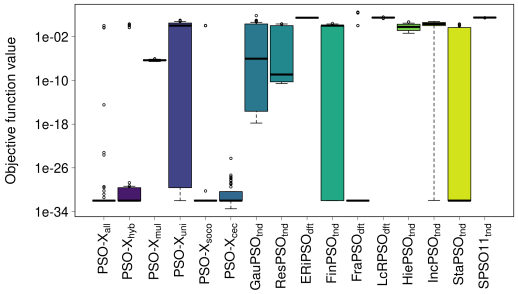
(f_7) Shifted Scfewels12 noise in fitness - CEC'05



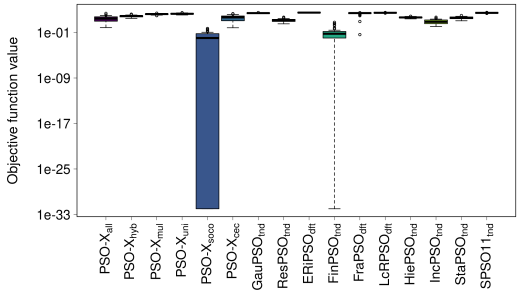
(f_8) Shifted Schwefel 2.22 - SOCO'10



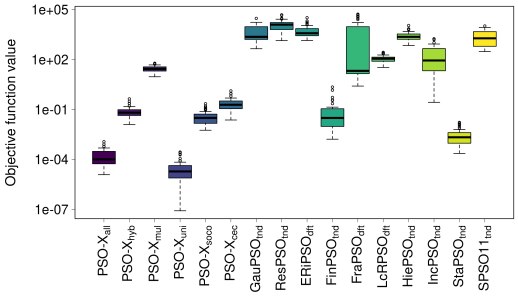
(f_9) Shifted Extended - SOCO'10



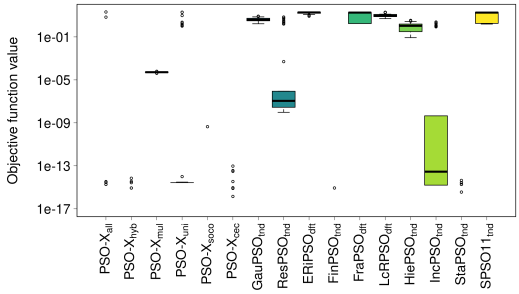
(f_{10}) Shifted Bohachevsky - SOCO'10



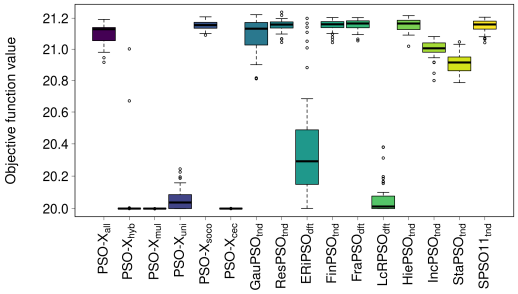
(f_{11}) Shifted Schaffer - CEC'05



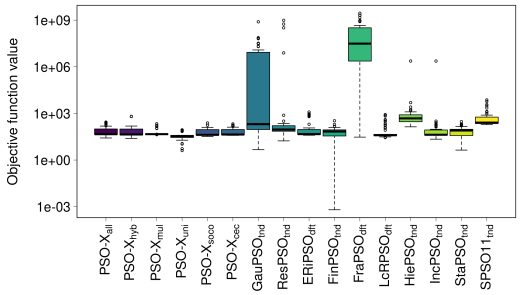
(f_{12}) Shchwefel 2.6 Global Optimum on Bounds - CEC'05



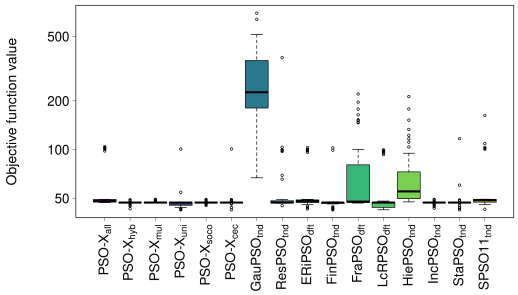
(f_{13}) Shifted Ackley - SOCO'10



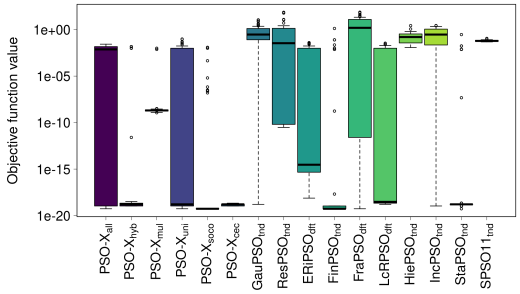
(f_{14}) Shifted Rotated Ackley - CEC'14



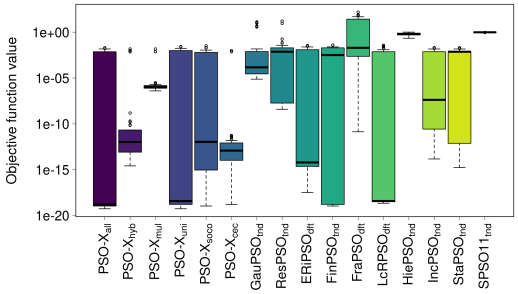
(f_{15}) Shifted Rosenbrock - SOCO'10



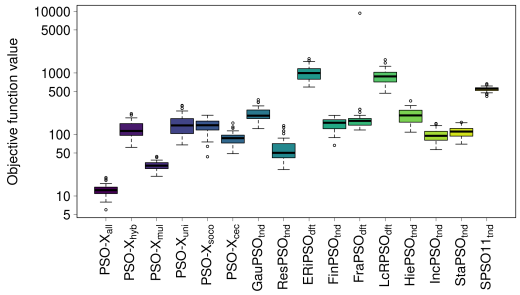
(f_{16}) Shifted Rotated Rosenbrock - CEC'14



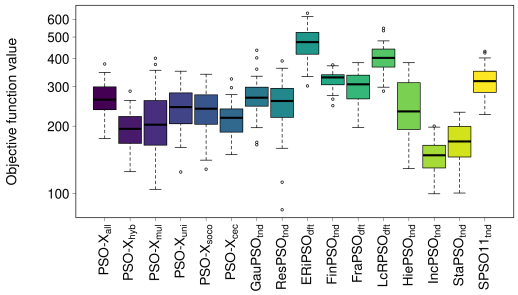
(f_{17}) Shifted Griewank - SOCO'10



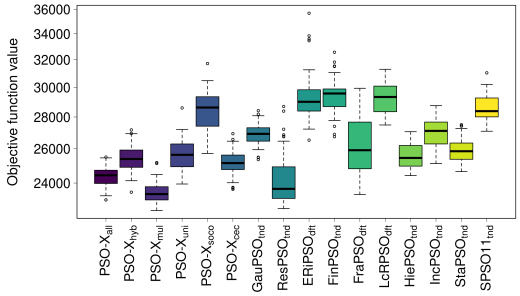
(f_{18}) Shifted Rotated Griewank - CEC'14



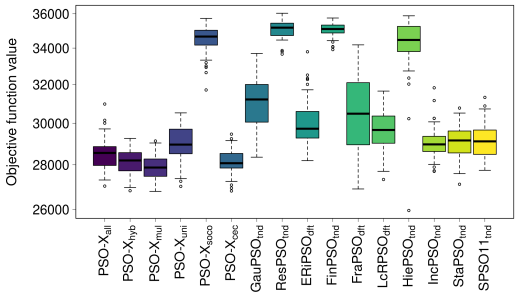
(f_{19}) Shifted Rastrigin - SOCO'10



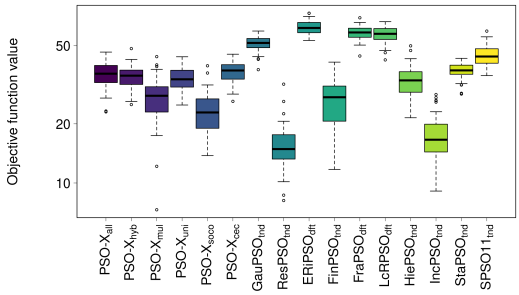
(f_{20}) Shifted Rotated Rastrigin - CEC'14



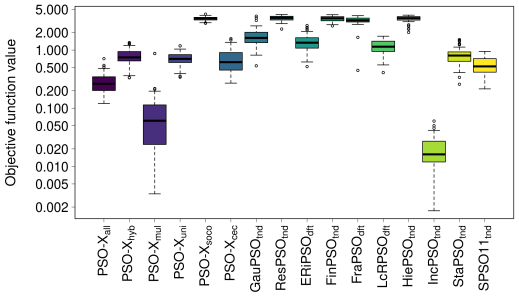
(f_{21}) Shifted Schwefel - SOCO'10



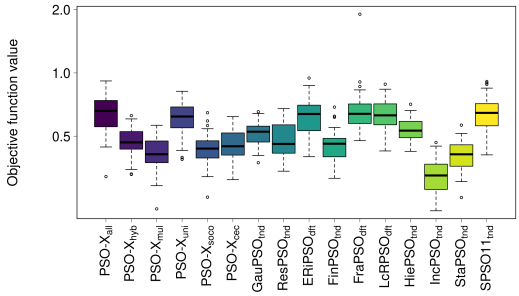
(f_{22}) Shifted Rotated Schwefel - CEC'14



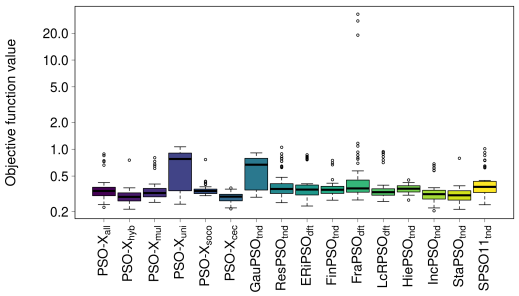
(f_{23}) Shifted Rotated WeierStrass - CEC'05



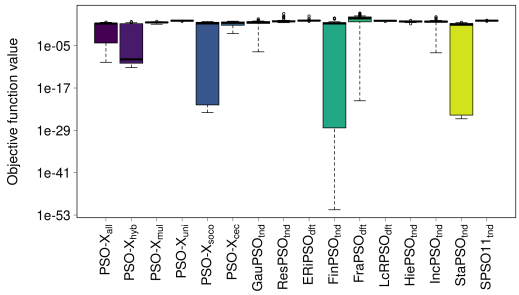
(f_{24}) Shifted Rotated Katsuura - CEC'14



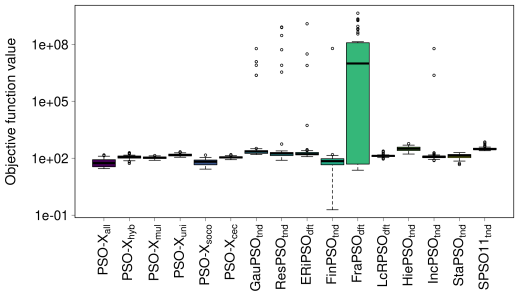
(f_{25}) Shifted Rotated HappyCat - CEC'14



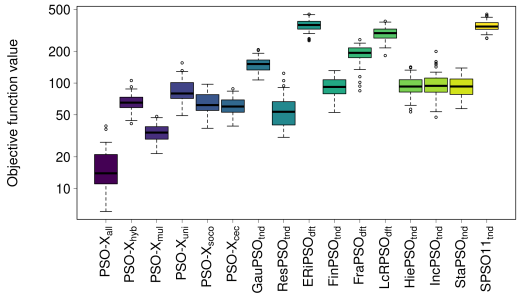
(f_{26}) Shifted Rotated HGBat - CEC'14



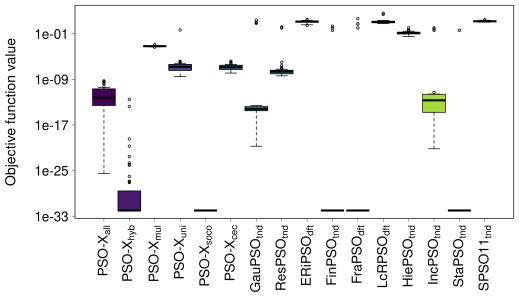
(f_{27}) Hybrid Function 1 (N = 2) - SOCO'10



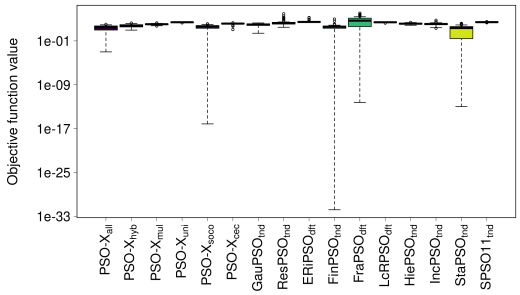
(f_{28}) Hybrid Function 2 (N = 2) - SOCO'10



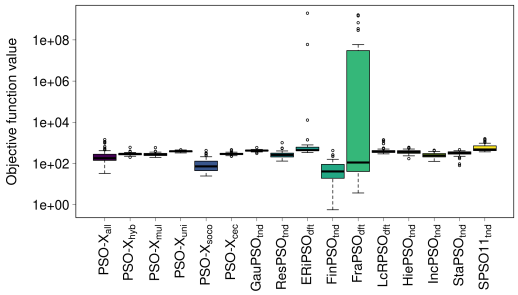
(f_{29}) Hybrid Function 3 (N = 2) - SOCO'10



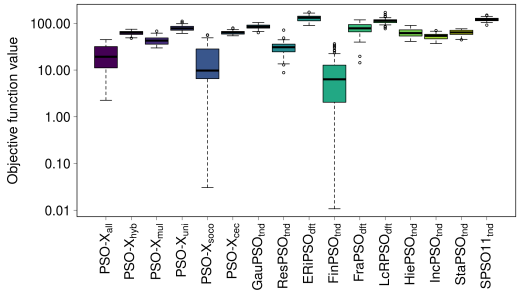
(f_{30}) Hybrid Function 4 (N = 2) - SOCO'10



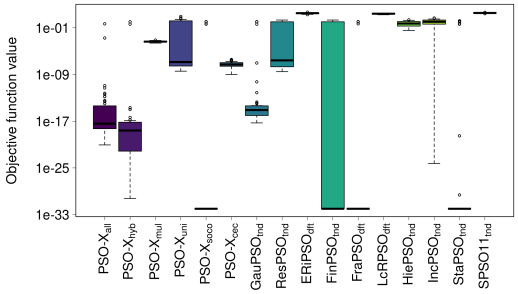
(f_{31}) Hybrid Function 7 (N = 2) - SOCO'10



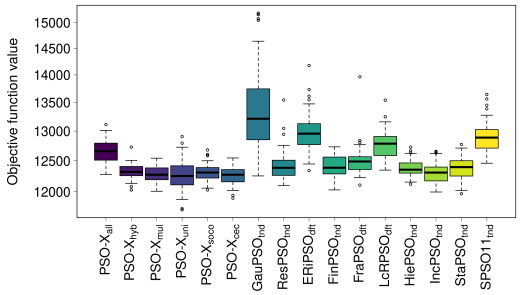
(f_{32}) Hybrid Function 8 (N = 2) - SOCO'10



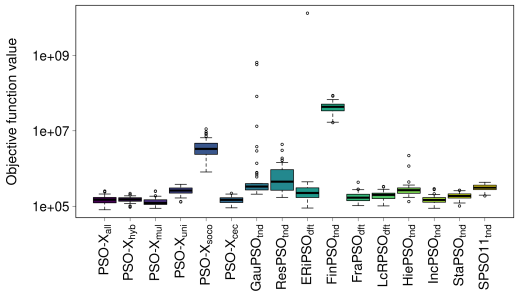
(f_{33}) Hybrid Function 9 (N = 2) - SOCO'10



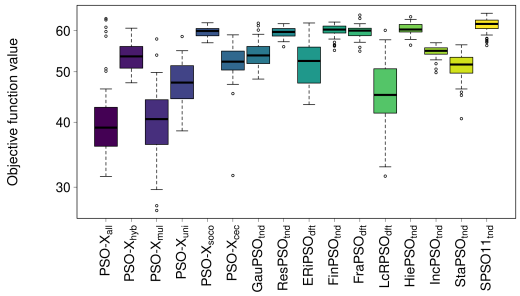
(f_{34}) Hybrid Function 10 (N = 2) - SOCO'10



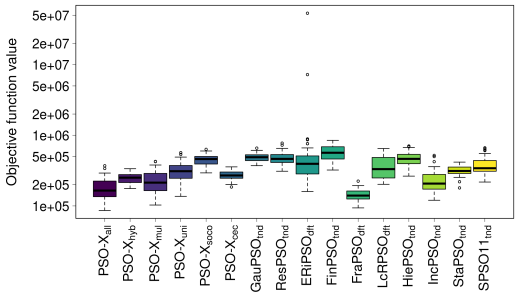
(f_{35}) Hybrid Function 1 (N = 3) - CEC'14



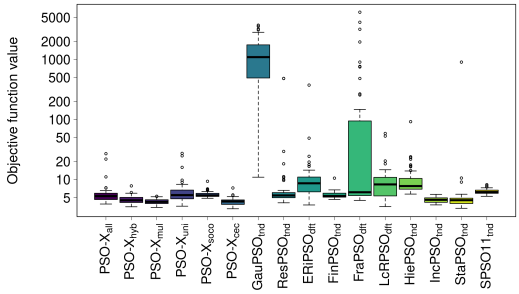
(f_{36}) Hybrid Function 2 (N = 3) - CEC'14



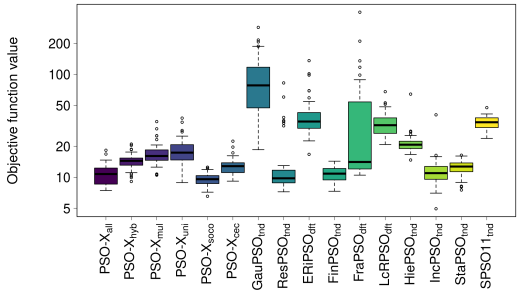
(f_{37}) Hybrid Function 3 (N = 4) - CEC'14



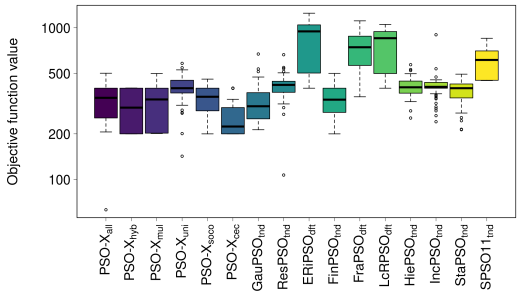
(f_{38}) Hybrid Function 4 (N = 4) - CEC'14



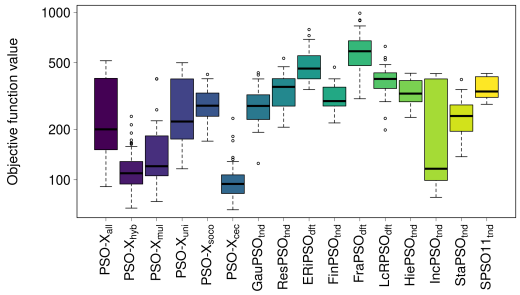
(f_{39}) Hybrid Function 5 (N = 5) - CEC'14



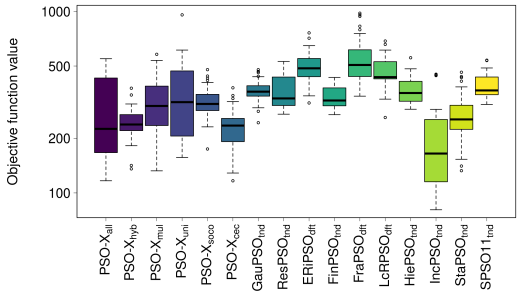
(f_{40}) Hybrid Function 6 (N = 5) - CEC'14



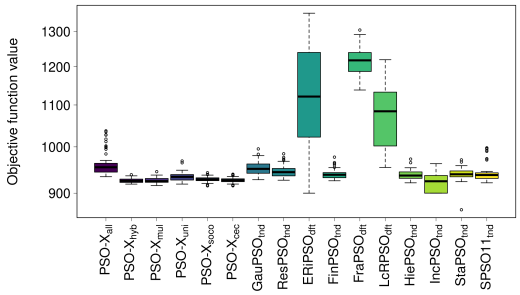
(f_{41}) Hybrid Composition Function - CEC'05



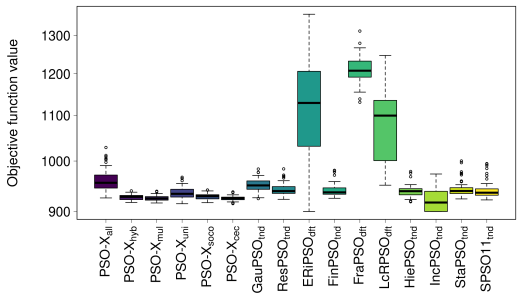
(f_{42}) Rotated Hybrid Composition Function - CEC'05



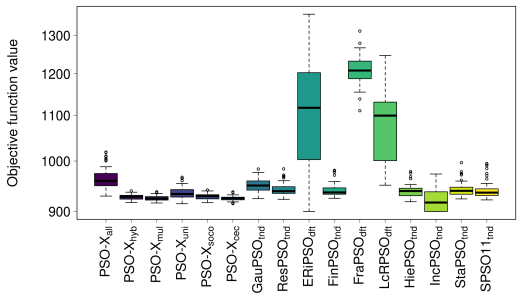
(f_{43}) Rotated H. Composition F. with Noise in Fitness - CEC'05



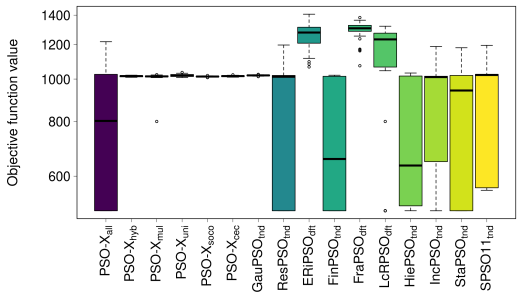
(f_{44}) Rotated Hybrid Composition F. - CEC'05



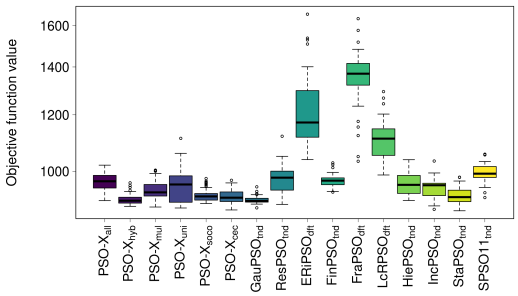
(f_{45}) Rotated H. Composition F. with a Narrow Basin for the Global Opt. - CEC'05



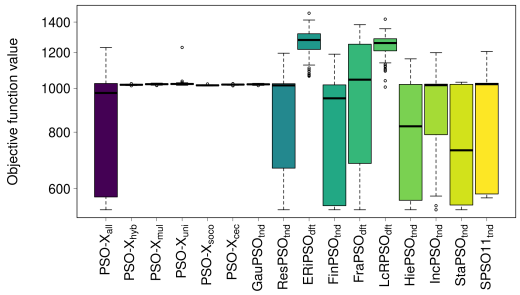
(f_{46}) Rotated H. Comp. F. with the Global Opt. On the Bounds - CEC'05



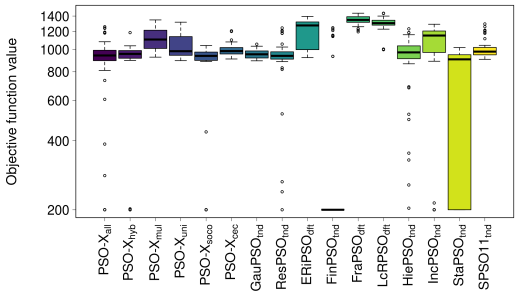
(f_{47}) Rotated Hybrid Composition Function - CEC'05



(f_{48}) Rotated H. Comp. F. with High Condition Number Matrix - CEC'05

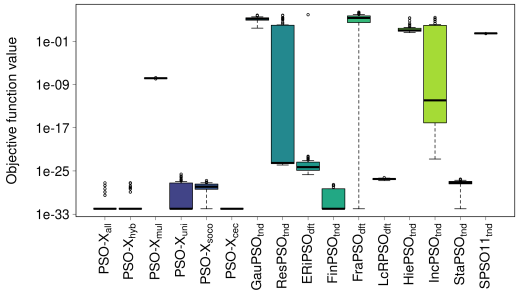


(f_{49}) Non-Continuous Rotated Hybrid Composition Function - CEC'05

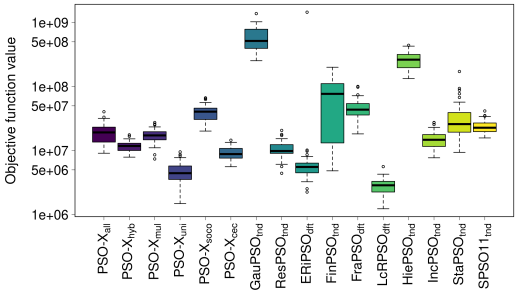


(f_{50}) Rotated Hybrid Composition Function - CEC'05

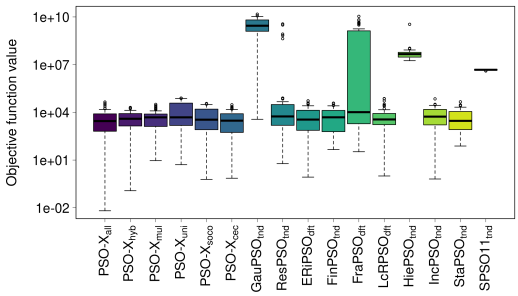
100 dimensions



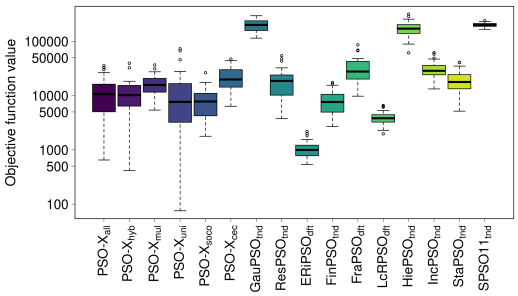
(f_1) Shifted Sphere - SOCO'10



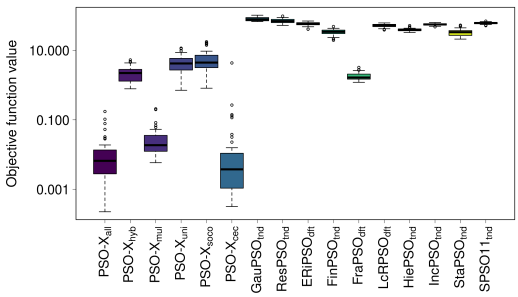
(f_2) Shifted Rotated High Conditioned Elliptic-CEC'14



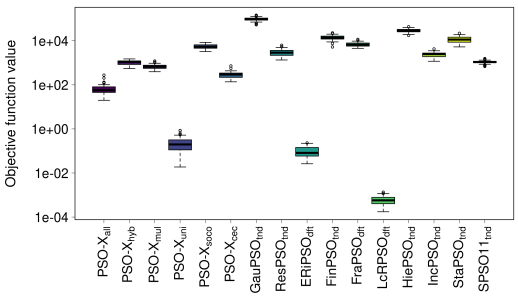
(f_3) Shifted Rotated Bent Cigar - CEC'14



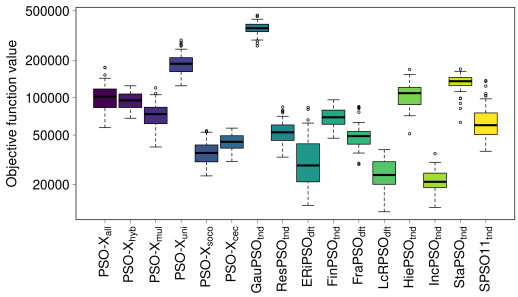
(f_4) Shifted Rotated Discus - CEC'14



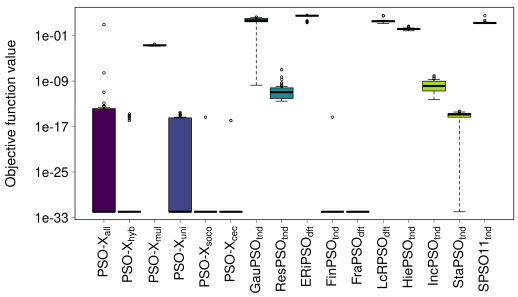
(f_5) Shifted Schwefel 22.1 - SOCO'10



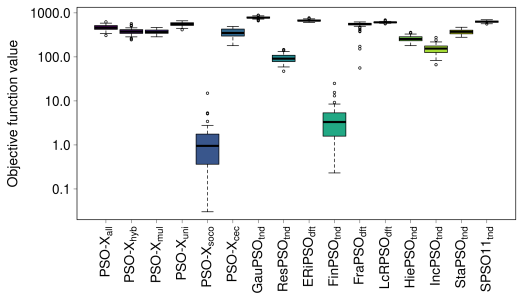
(f_6) Rotated Schwefel 1.2 - SOCO'10



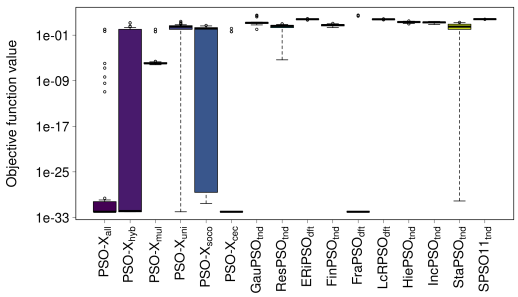
(f_7) Shifted Scfewels12 noise in fitness - CEC'05



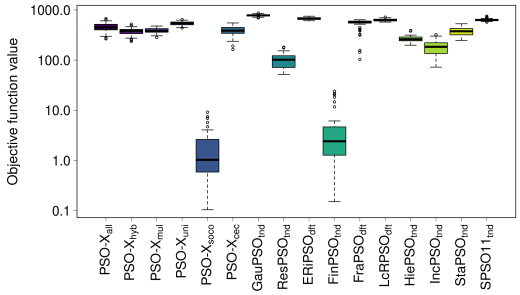
(f_8) Shifted Schwefel 2.22 - SOCO'10



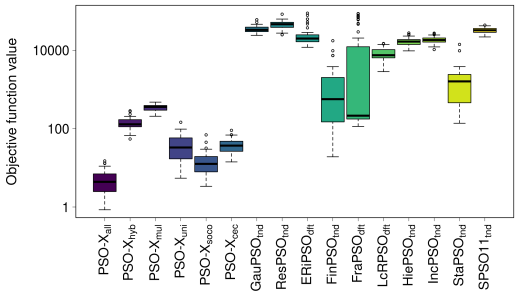
(f_9) Shifted Extended - SOCO'10



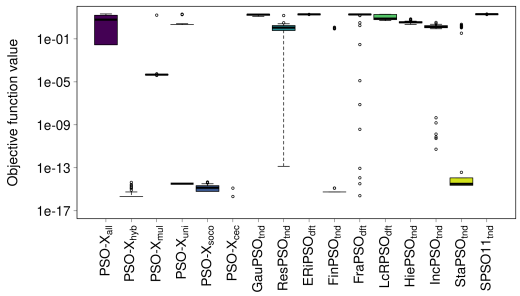
(f_{10}) Shifted Bohachevsky - SOCO'10



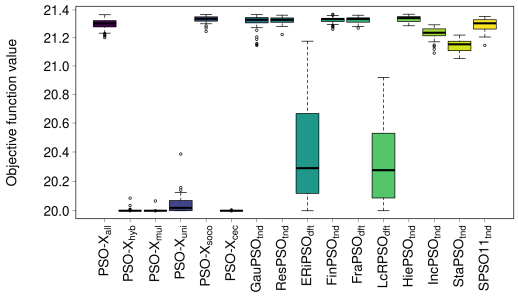
(f_{11}) Shifted Schaffer - CEC'05



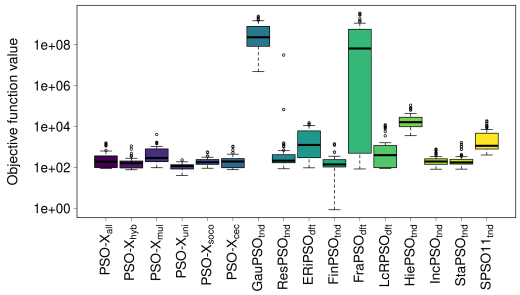
(f_{12}) Shchwefel 2.6 Global Optimum on Bounds - CEC'05



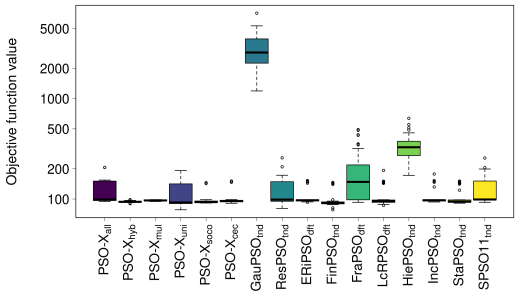
(f_{13}) Shifted Ackley - SOCO'10



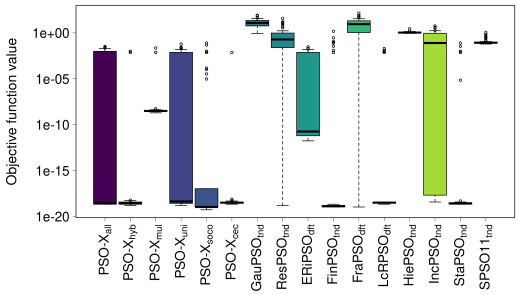
(f_{14}) Shifted Rotated Ackley - CEC'14



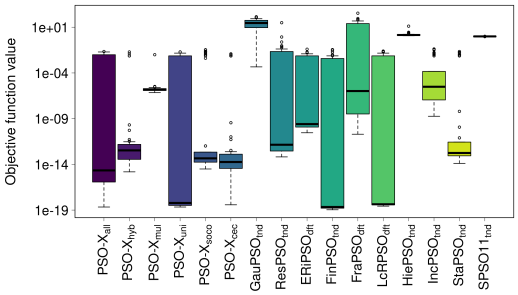
(f_{15}) Shifted Rosenbrock - SOCO'10



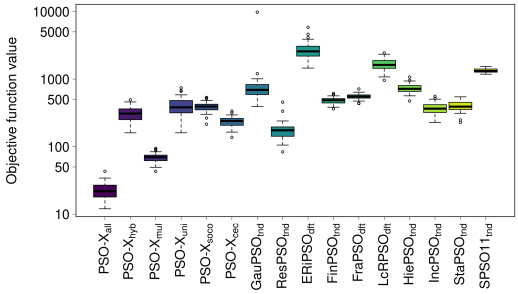
(f_{16}) Shifted Rotated Rosenbrock - CEC'14



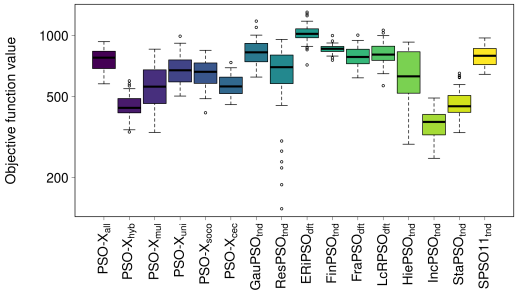
(f_{17}) Shifted Griewank - SOCO'10



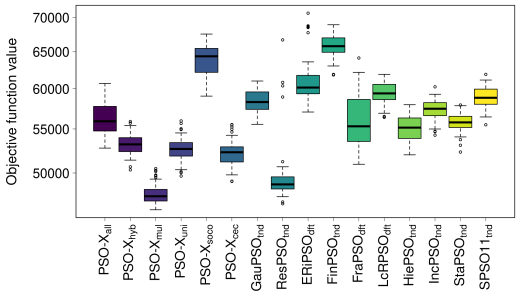
(f_{18}) Shifted Rotated Griewank - CEC'14



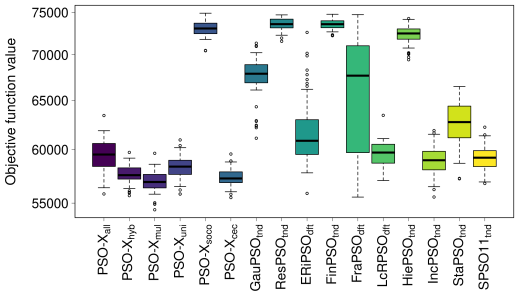
(f_{19}) Shifted Rastrigin - SOCO'10



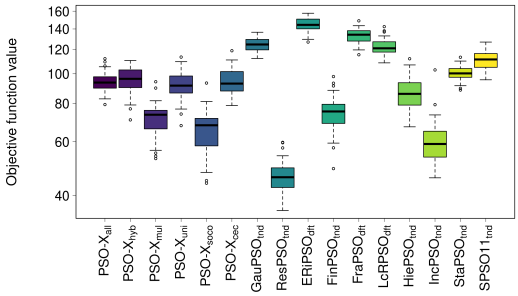
(f_{20}) Shifted Rotated Rastrigin - CEC'14



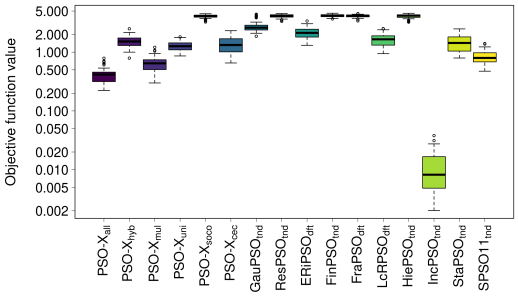
(f_{21}) Shifted Schwefel - SOCO'10



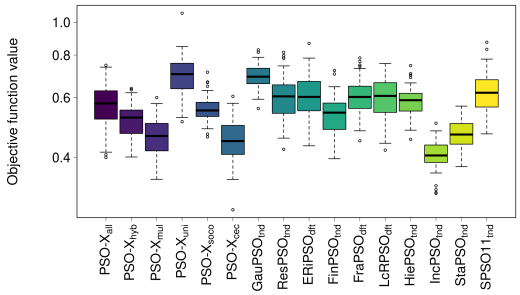
(f_{22}) Shifted Rotated Schwefel - CEC'14



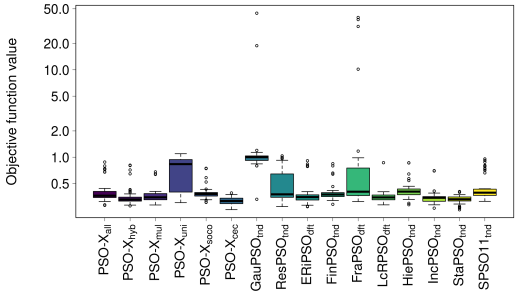
(f_{23}) Shifted Rotated WeierStrass - CEC'05



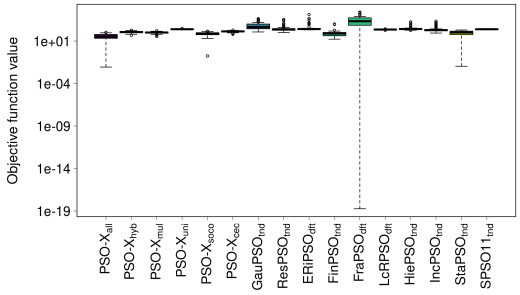
(f_{24}) Shifted Rotated Katsuura - CEC'14



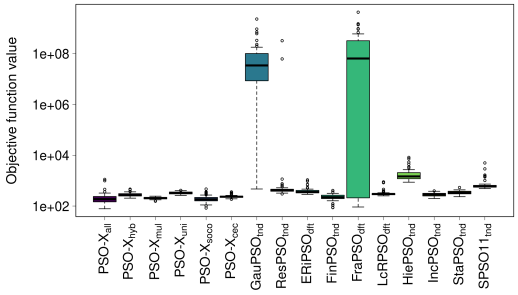
(f_{25}) Shifted Rotated HappyCat - CEC'14



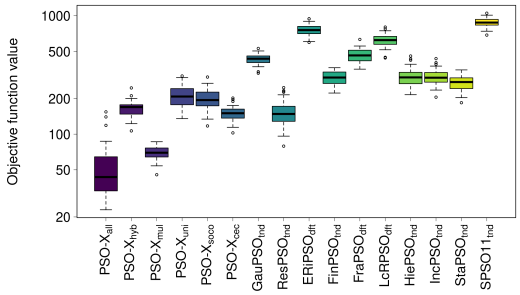
(f_{26}) Shifted Rotated HGBat - CEC'14



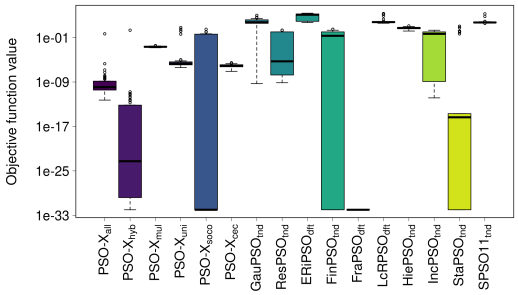
(f_{27}) Hybrid Function 1 (N = 2) - SOCO'10



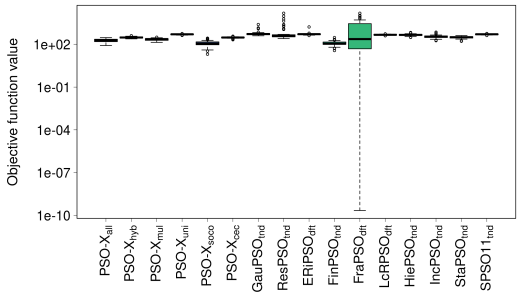
(f_{28}) Hybrid Function 2 (N = 2) - SOCO'10



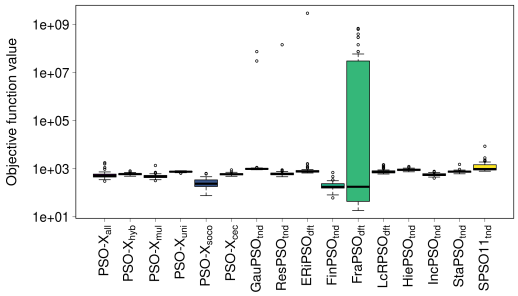
(f_{29}) Hybrid Function 3 (N = 2) - SOCO'10



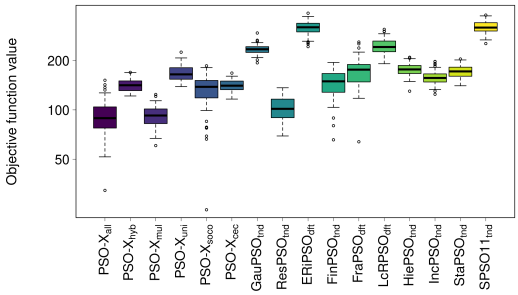
(f_{30}) Hybrid Function 4 (N = 2) - SOCO'10



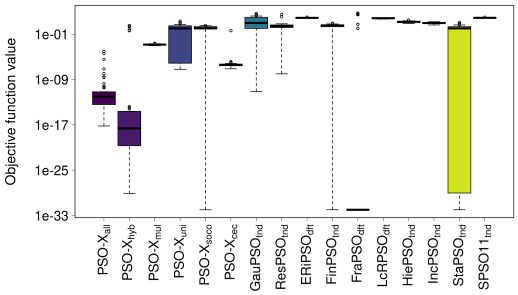
(f_{31}) Hybrid Function 7 (N = 2) - SOCO'10



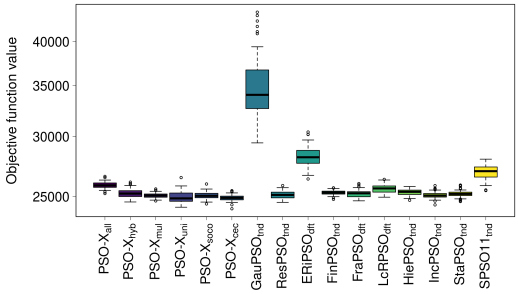
(f_{32}) Hybrid Function 8 (N = 2) - SOCO'10



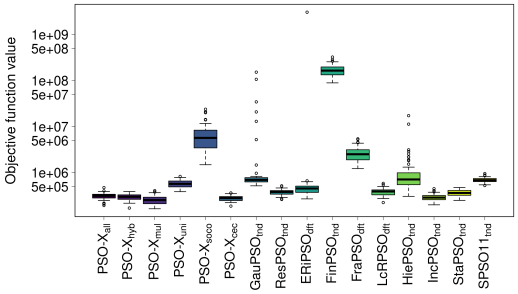
(f_{33}) Hybrid Function 9 (N = 2) - SOCO'10



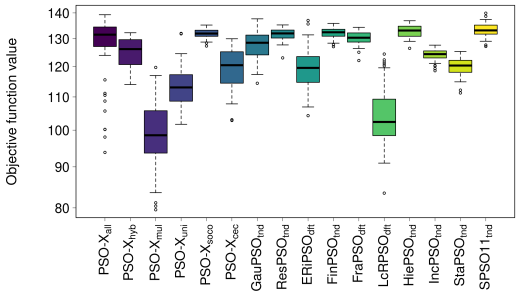
(f_{34}) Hybrid Function 10 (N = 2) - SOCO'10



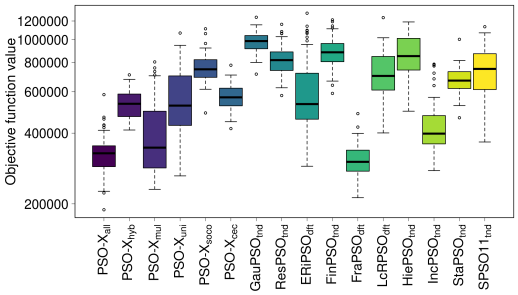
(f_{35}) Hybrid Function 1 (N = 3) - CEC'14



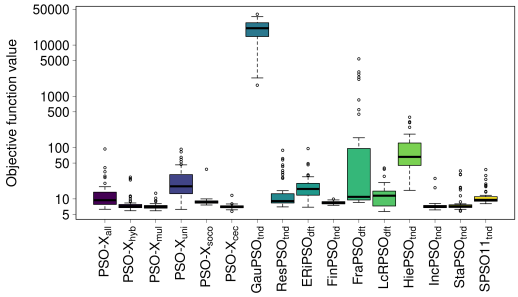
(f_{36}) Hybrid Function 2 (N = 3) - CEC'14



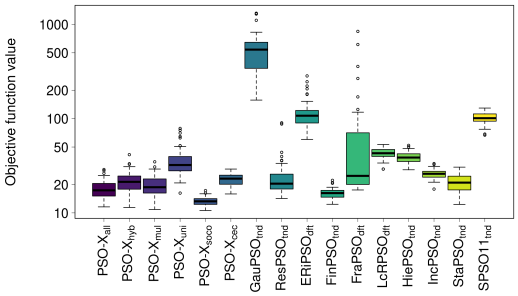
(f_{37}) Hybrid Function 3 (N = 4) - CEC'14



(f_{38}) Hybrid Function 4 (N = 4) - CEC'14



(f_{39}) Hybrid Function 5 (N = 5) - CEC'14

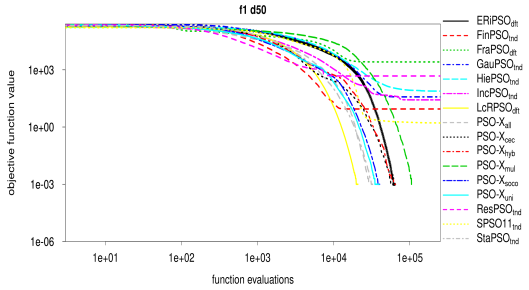


(f_{40}) Hybrid Function 6 (N = 5) - CEC'14

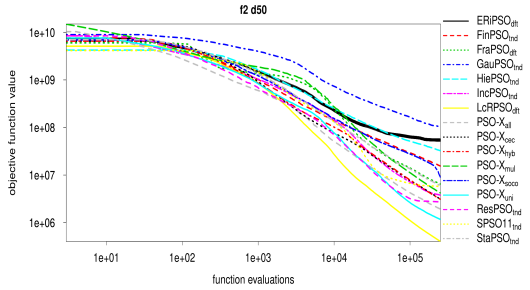
B.5 Convergence Plots of the 16 Compared PSO Algorithms

In this section, we report the convergence plots of the 16 PSO algorithms. Each of this plots shows the relation between the best solution quality and the number of function evaluations used by the algorithms. In all cases, the algorithm was stopped when it reached $5\,000 \cdot d$ function evaluations. We present a total of 90 plots: the first 50 correspond to the functions with $d = 50$ and the remaining 40 to those with $d = 100$.

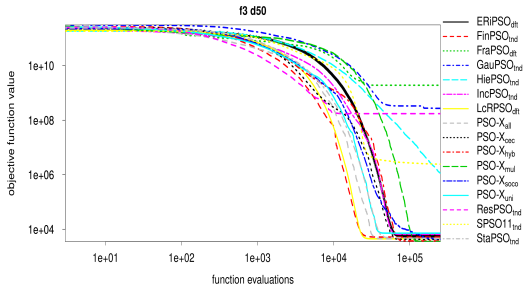
50 dimensions



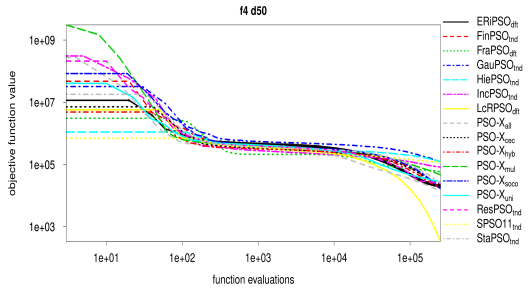
(f_1) Shifted Sphere - SOCO'10



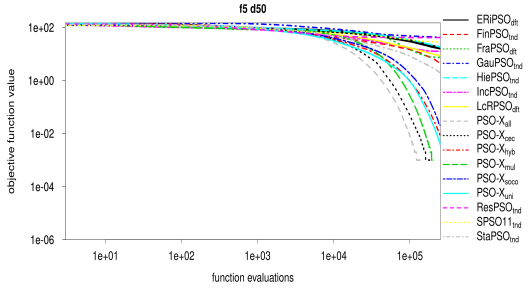
(f_2) Shifted Rotated High Conditioned Elliptic-CEC'14



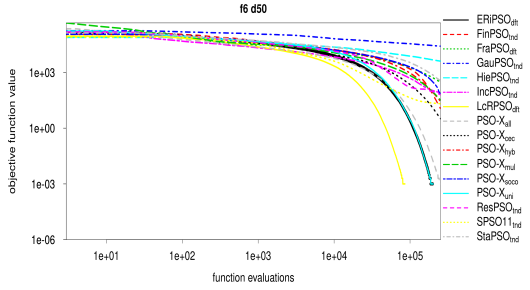
(f_3) Shifted Rotated Bent Cigar - CEC'14



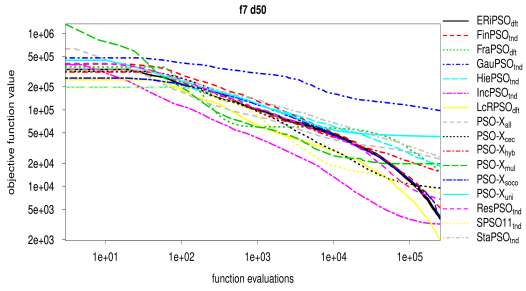
(f_4) Shifted Rotated Discus - CEC'14



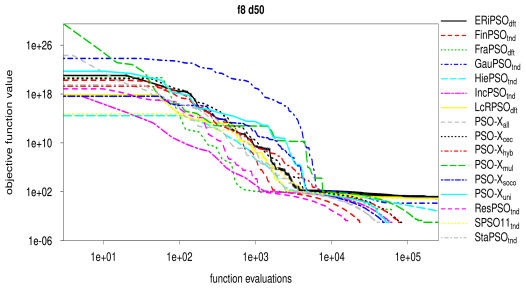
(f_5) Shifted Schwefel 22.1 - SOCO'10



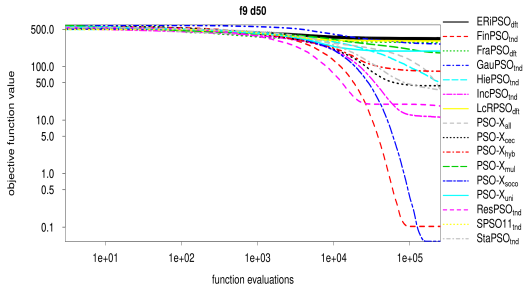
(f_6) Rotated Schwefel 1.2 - SOCO'10



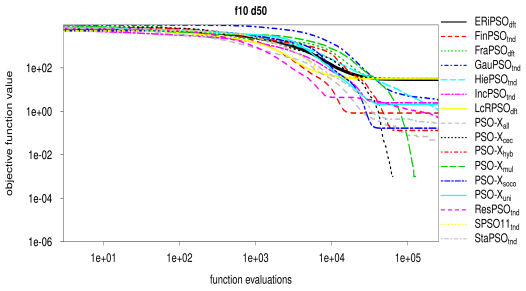
(f_7) Shifted Scfewels12 noise in fitness - CEC'05



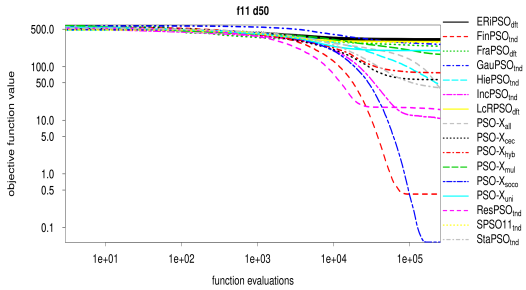
(f_8) Shifted Schwefel 2.22 - SOCO'10



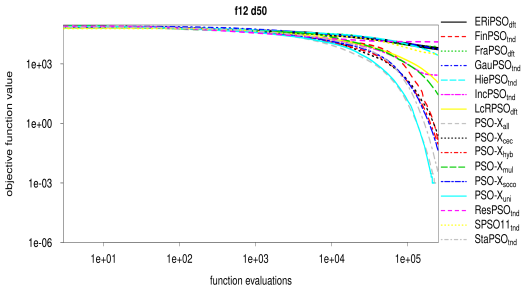
(f_9) Shifted Extended - SOCO'10



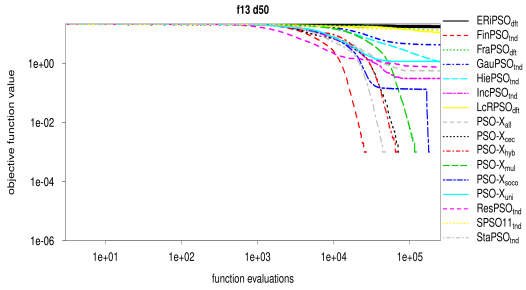
(f_{10}) Shifted Bohachevsky - SOCO'10



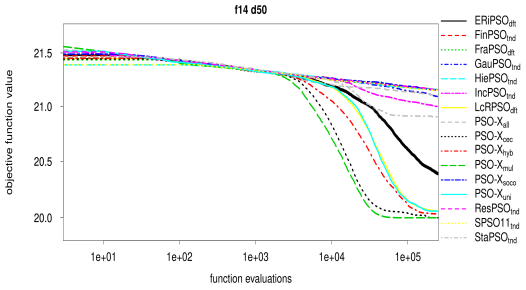
(f_{11}) Shifted Schaffer - CEC'05



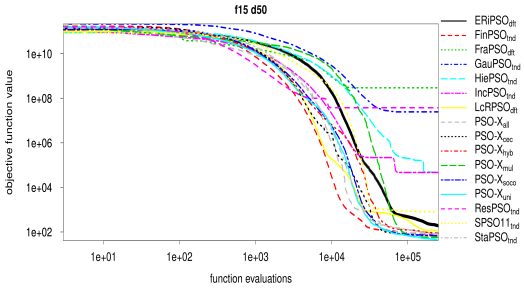
(f_{12}) Shchwefel 2.6 Global Optimum on Bounds - CEC'05



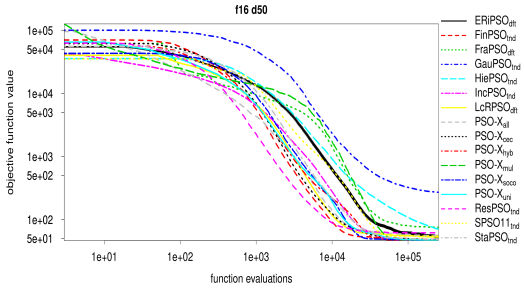
(f_{13}) Shifted Ackley - SOCO'10



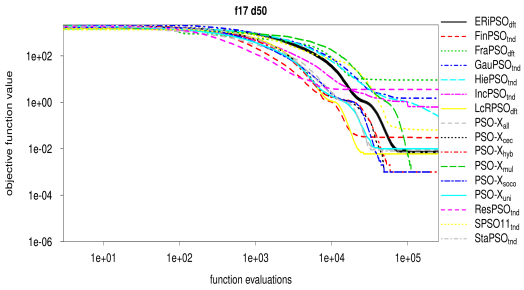
(f_{14}) Shifted Rotated Ackley - CEC'14



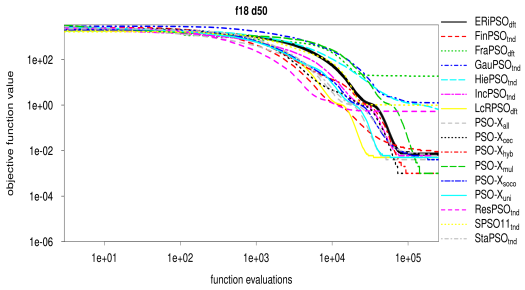
(f_{15}) Shifted Rosenbrock - SOCO'10



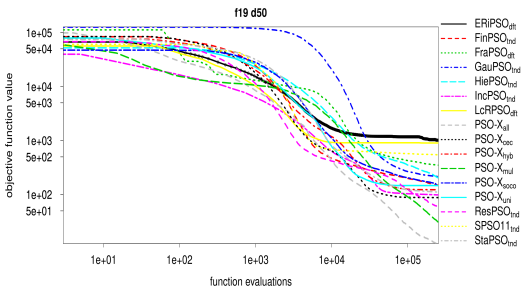
(f_{16}) Shifted Rotated Rosenbrock - CEC'14



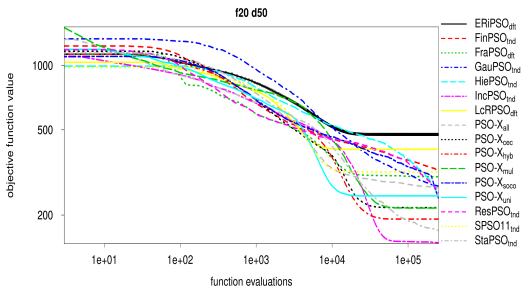
(f_{17}) Shifted Griewank - SOCO'10



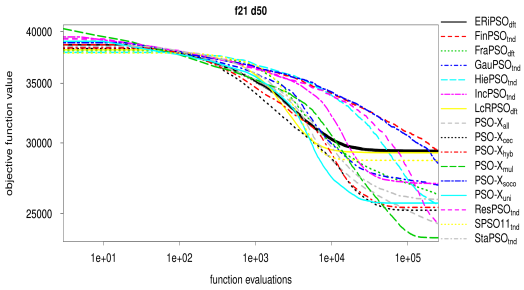
(f_{18}) Shifted Rotated Griewank - CEC'14



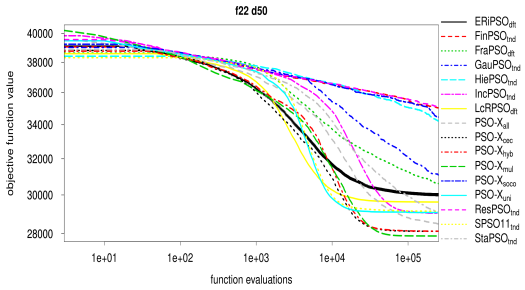
(f_{19}) Shifted Rastrigin - SOCO'10



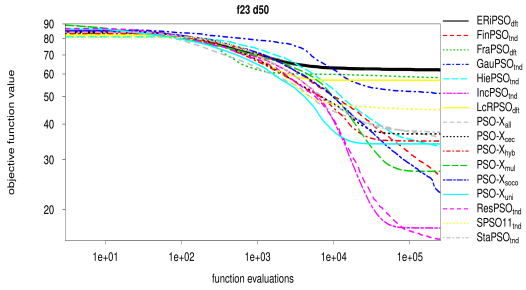
(f_{20}) Shifted Rotated Rastrigin - CEC'14



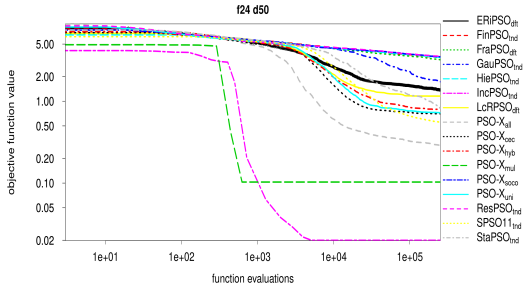
(f_{21}) Shifted Schwefel - SOCO'10



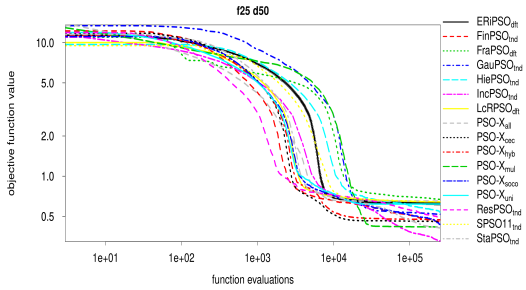
(f_{22}) Shifted Rotated Schwefel - CEC'14



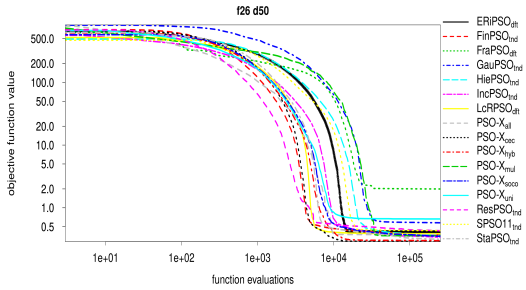
(f_{23}) Shifted Rotated WeierStrass - CEC'05



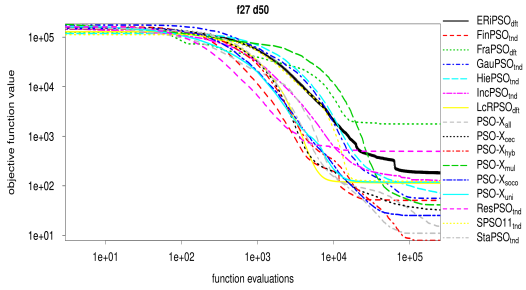
(f_{24}) Shifted Rotated Katsuura - CEC'14



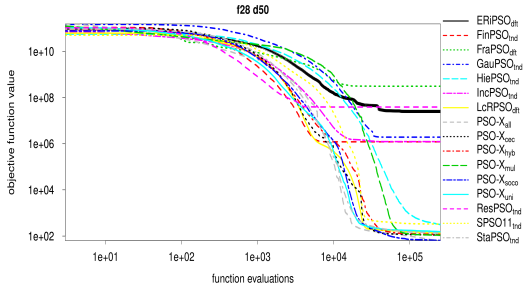
(f_{25}) Shifted Rotated HappyCat - CEC'14



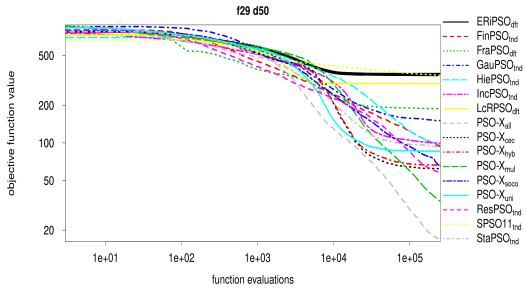
(f_{26}) Shifted Rotated HGBat - CEC'14



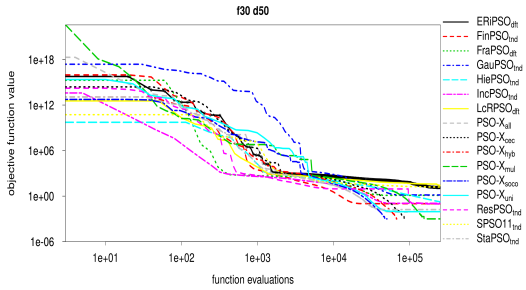
(f_{27}) Hybrid Function 1 (N = 2) - SOCO'10



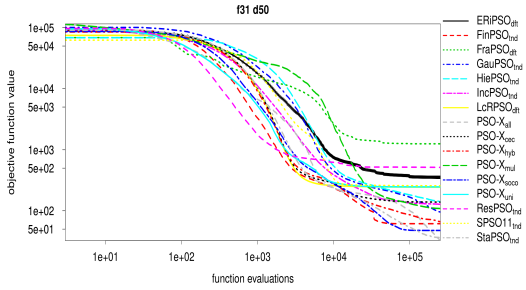
(f_{28}) Hybrid Function 2 (N = 2) - SOCO'10



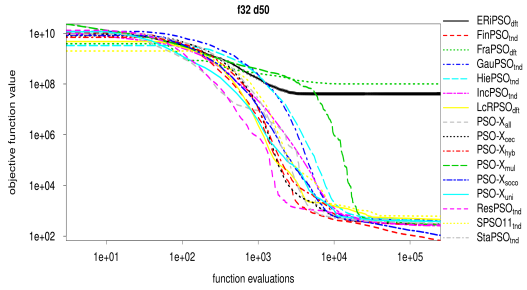
(f_{29}) Hybrid Function 3 (N = 2) - SOCO'10



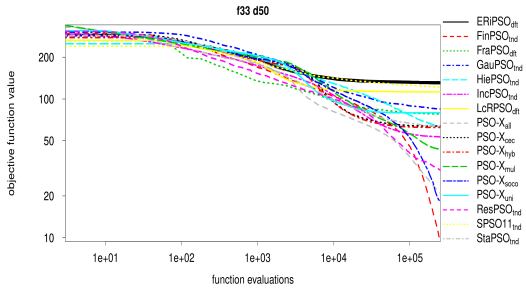
(f_{30}) Hybrid Function 4 (N = 2) - SOCO'10



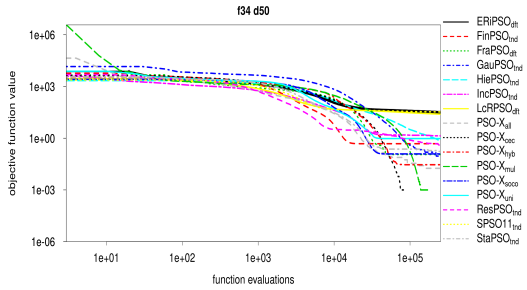
(f_{31}) Hybrid Function 7 (N = 2) - SOCO'10



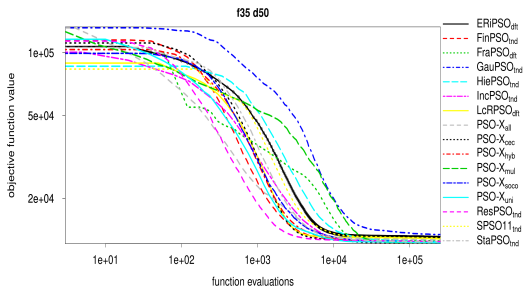
(f_{32}) Hybrid Function 8 (N = 2) - SOCO'10



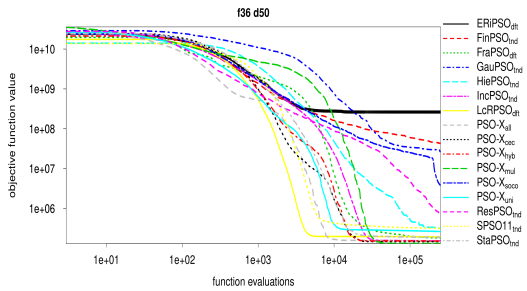
(f_{33}) Hybrid Function 9 (N = 2) - SOCO'10



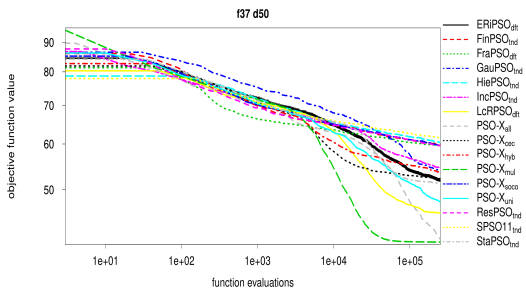
(f_{34}) Hybrid Function 10 (N = 2) - SOCO'10



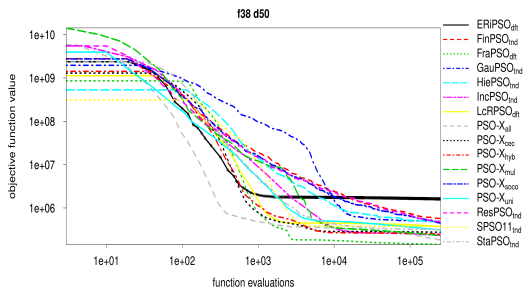
(f_{35}) Hybrid Function 1 (N = 3) - CEC'14



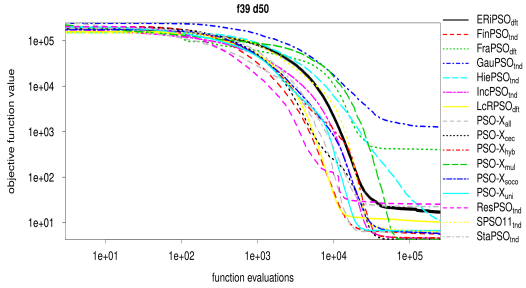
(f_{36}) Hybrid Function 2 (N = 3) - CEC'14



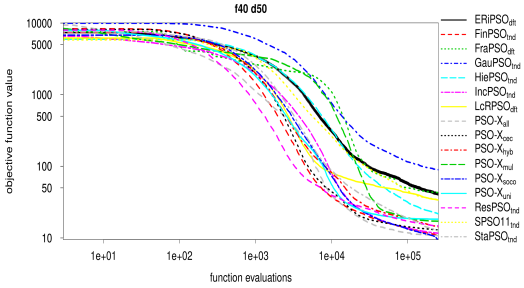
(f_{37}) Hybrid Function 3 (N = 4) - CEC'14



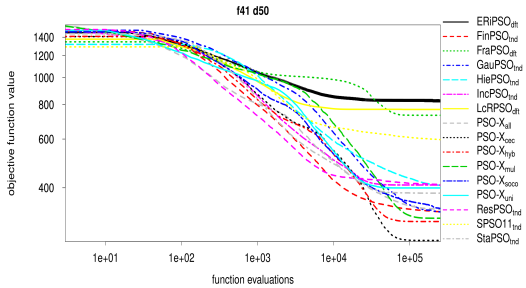
(f_{38}) Hybrid Function 4 (N = 4) - CEC'14



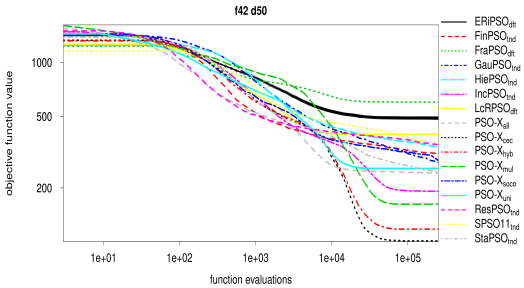
(f_{39}) Hybrid Function 5 (N = 5) - CEC'14



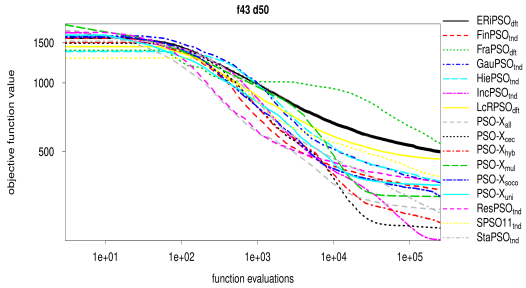
(f_{40}) Hybrid Function 6 (N = 5) - CEC'14



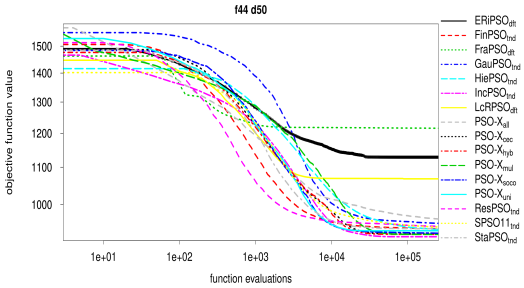
(f_{41}) Hybrid Composition Function - CEC'05



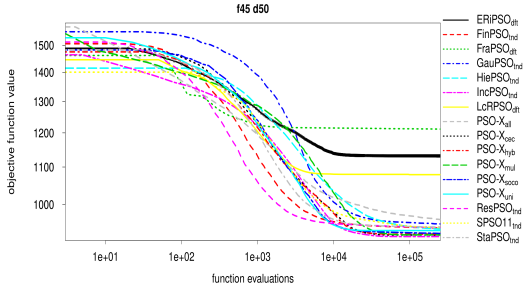
(f_{42}) Rotated Hybrid Composition Function - CEC'05



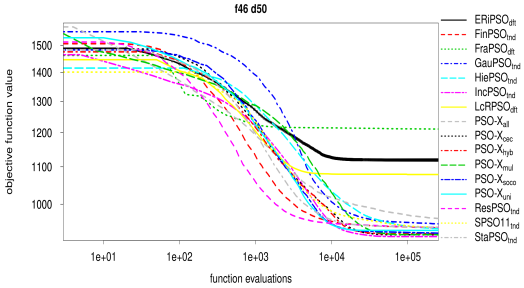
(f_{43}) Rotated H. Composition F. with Noise in Fitness - CEC'05



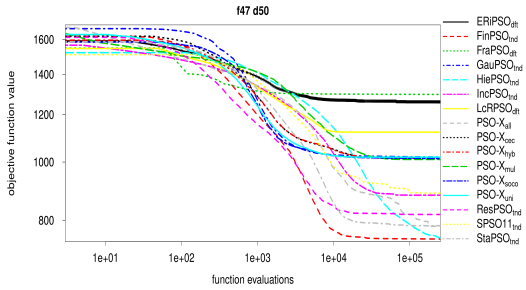
(f_{44}) Rotated Hybrid Composition F. - CEC'05



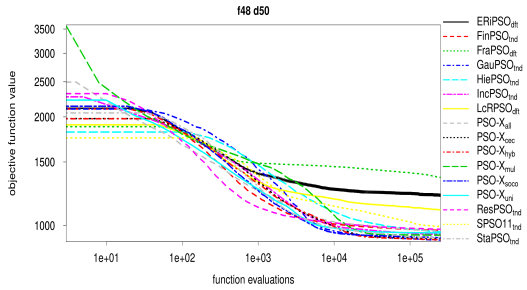
(f_{45}) Rotated H. Composition F. with a Narrow Basin for the Global Opt. - CEC'05



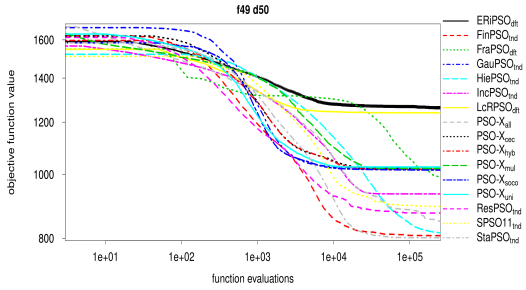
(f_{46}) Rotated H. Comp. F. with the Global Opt. On the Bounds - CEC'05



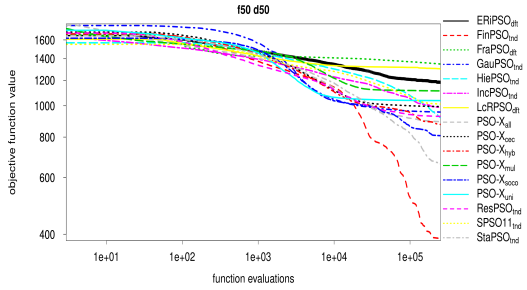
(f_{47}) Rotated Hybrid Composition Function - CEC'05



(f_{48}) Rotated H. Comp. F. with High Condition Number Matrix - CEC'05

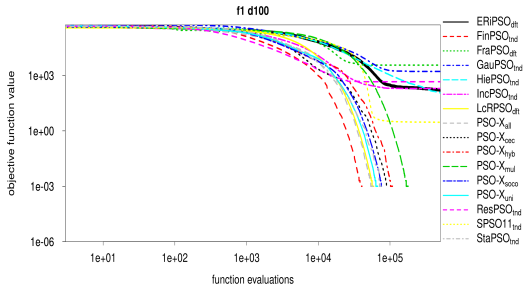


(f_{49}) Non-Continuous Rotated Hybrid Composition Function - CEC'05

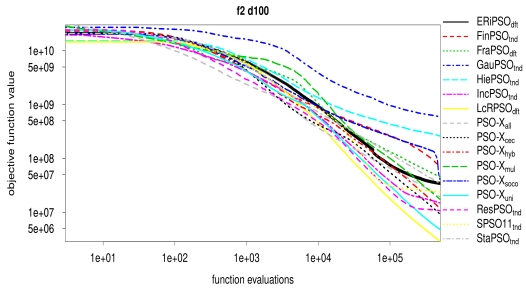


(f_{50}) Rotated Hybrid Composition Function - CEC'05

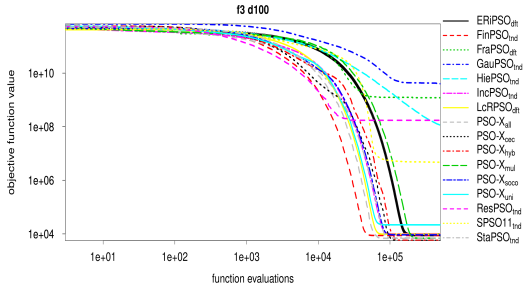
100 dimensions



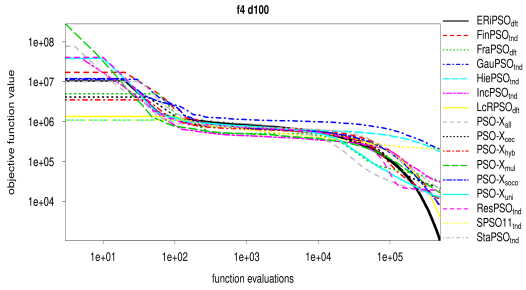
(f_1) Shifted Sphere - SOCO'10



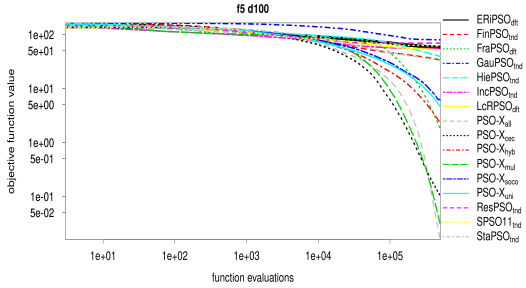
(f_2) Shifted Rotated High Conditioned Elliptic-CEC'14



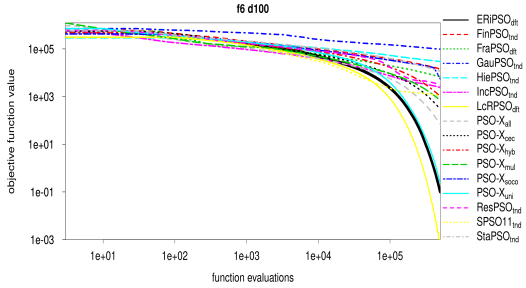
(f_3) Shifted Rotated Bent Cigar - CEC'14



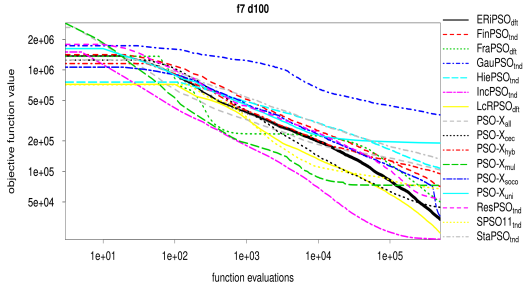
(f_4) Shifted Rotated Discus - CEC'14



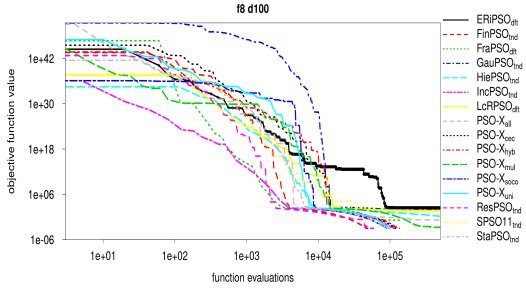
(f_5) Shifted Schwefel 22.1 - SOCO'10



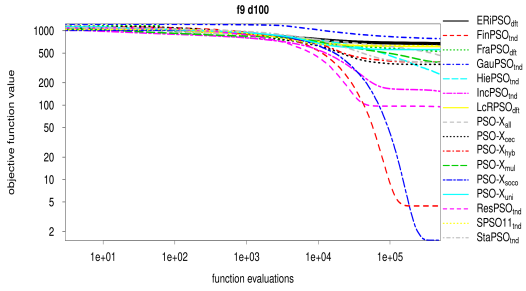
(f_6) Rotated Schwefel 1.2 - SOCO'10



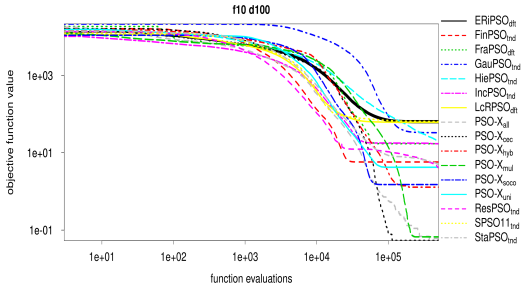
(f_7) Shifted Scfewels12 noise in fitness - CEC'05



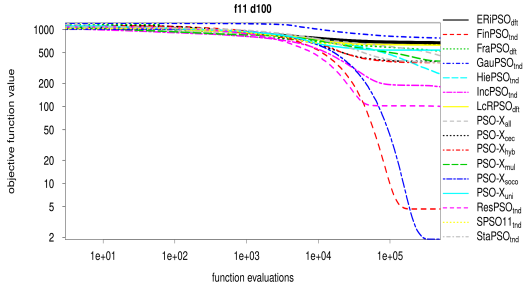
(f_8) Shifted Schwefel 2.22 - SOCO'10



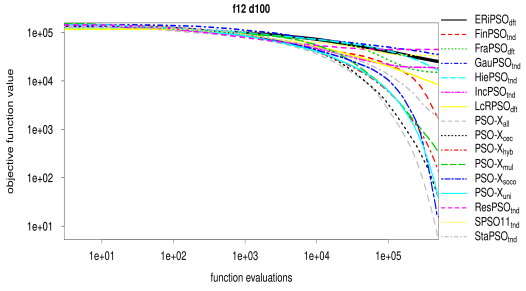
(f_9) Shifted Extended - SOCO'10



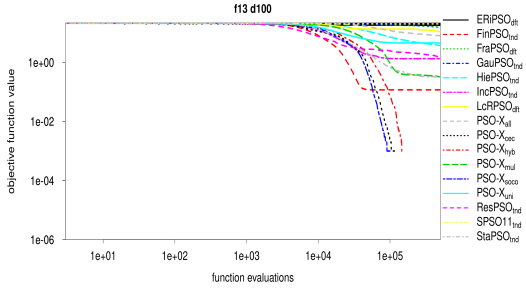
(f_{10}) Shifted Bohachevsky - SOCO'10



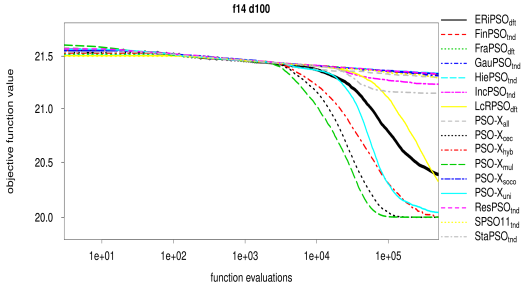
(f_{11}) Shifted Schaffer - CEC'05



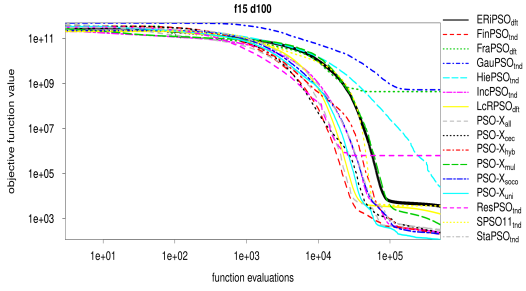
(f_{12}) Shchwefel 2.6 Global Optimum on Bounds - CEC'05



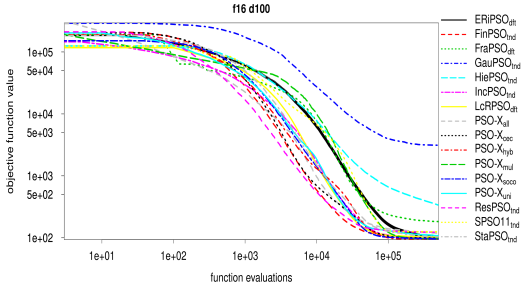
(f_{13}) Shifted Ackley - SOCO'10



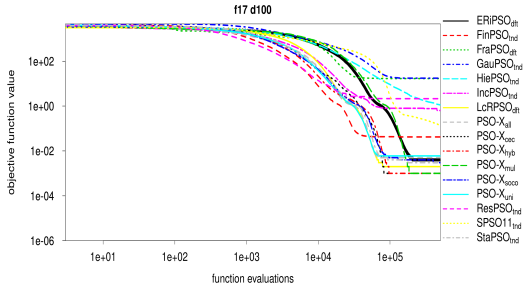
(f_{14}) Shifted Rotated Ackley - CEC'14



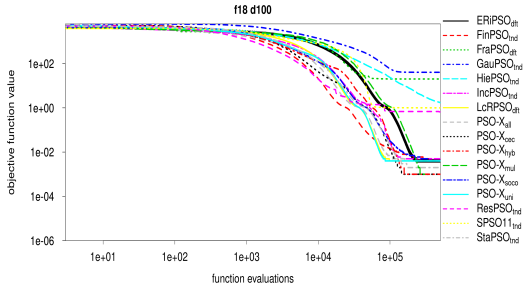
(f_{15}) Shifted Rosenbrock - SOCO'10



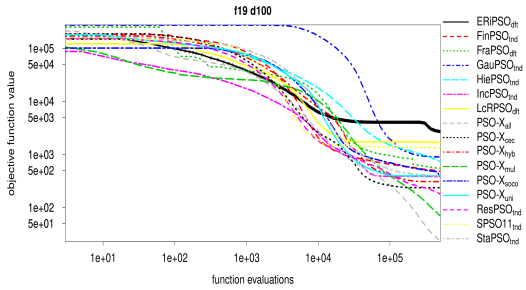
(f_{16}) Shifted Rotated Rosenbrock - CEC'14



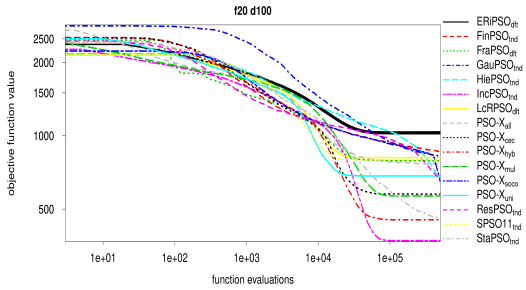
(f_{17}) Shifted Griewank - SOCO'10



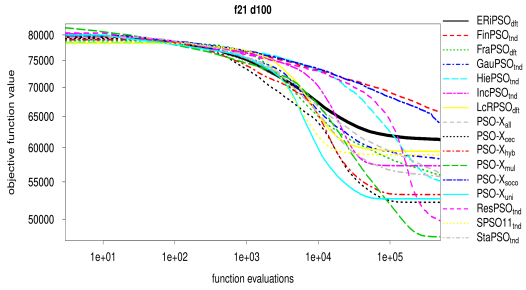
(f_{18}) Shifted Rotated Griewank - CEC'14



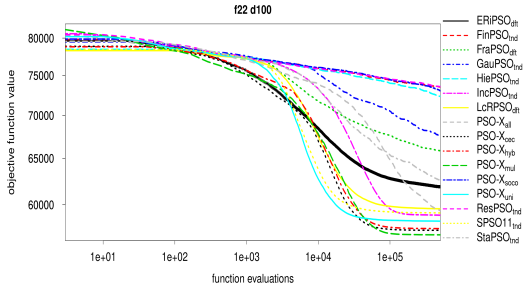
(f_{19}) Shifted Rastrigin - SOCO'10



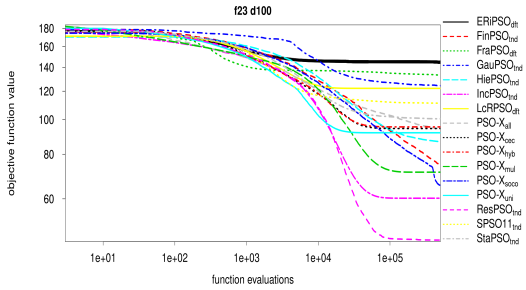
(f_{20}) Shifted Rotated Rastrigin - CEC'14



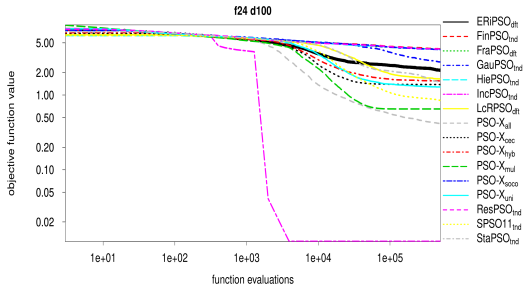
(f_{21}) Shifted Schwefel - SOCO'10



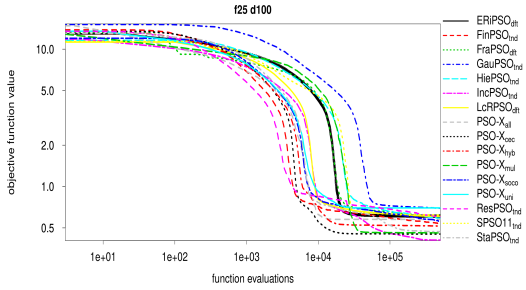
(f_{22}) Shifted Rotated Schwefel - CEC'14



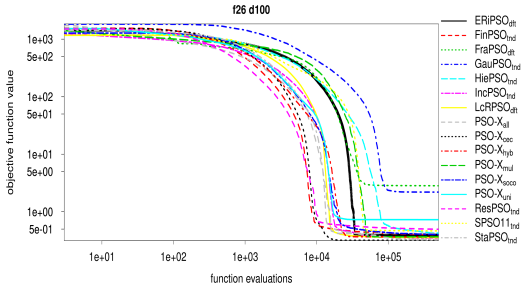
(f_{23}) Shifted Rotated WeierStrass - CEC'05



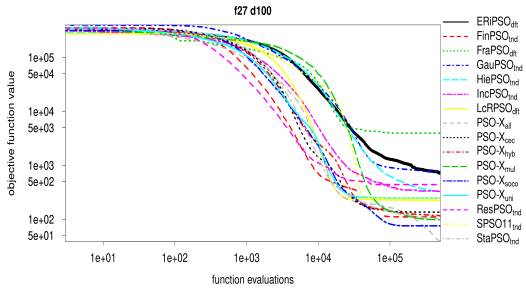
(f_{24}) Shifted Rotated Katsuura - CEC'14



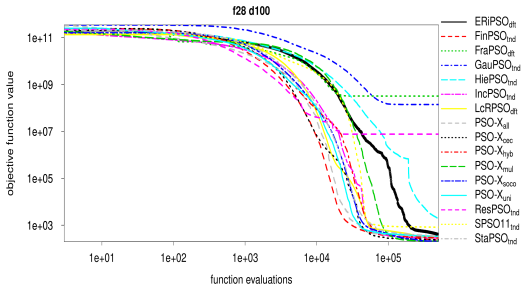
(f_{25}) Shifted Rotated HappyCat - CEC'14



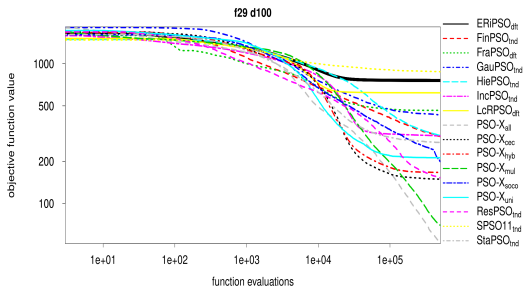
(f_{26}) Shifted Rotated HGBat - CEC'14



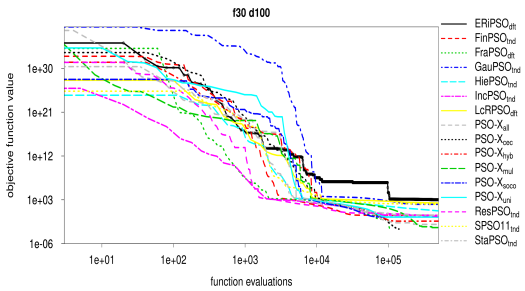
(f_{27}) Hybrid Function 1 (N = 2) - SOCO'10



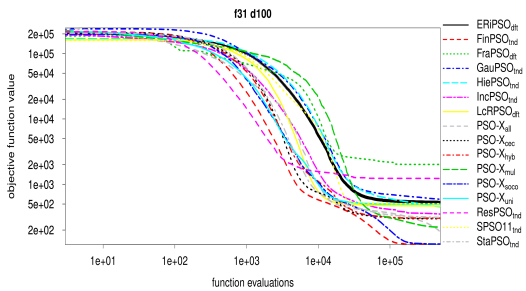
(f_{28}) Hybrid Function 2 (N = 2) - SOCO'10



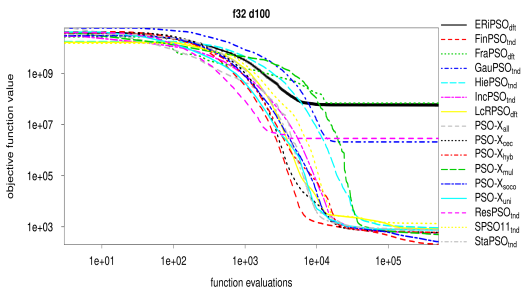
(f_{29}) Hybrid Function 3 (N = 2) - SOCO'10



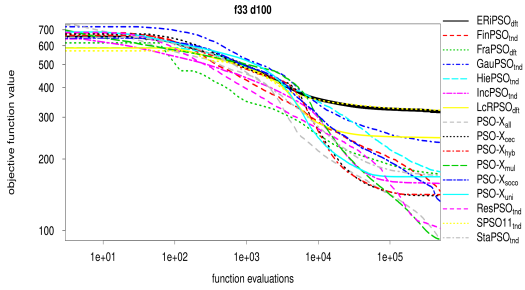
(f_{30}) Hybrid Function 4 (N = 2) - SOCO'10



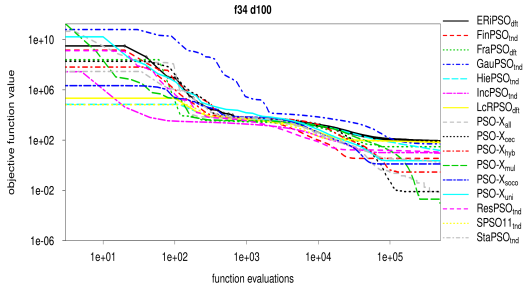
(f_{31}) Hybrid Function 7 (N = 2) - SOCO'10



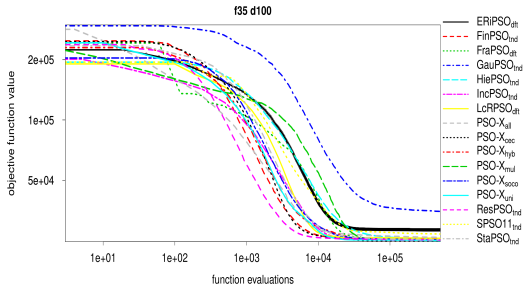
(f_{32}) Hybrid Function 8 (N = 2) - SOCO'10



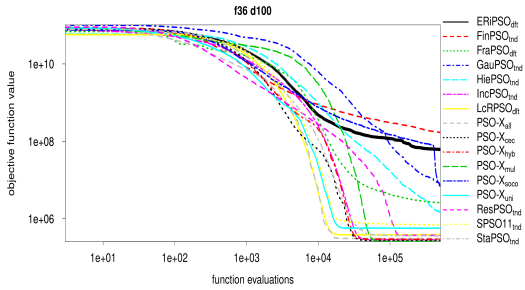
(f_{33}) Hybrid Function 9 (N = 2) - SOCO'10



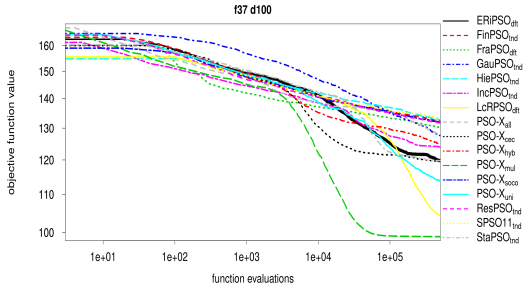
(f_{34}) Hybrid Function 10 (N = 2) - SOCO'10



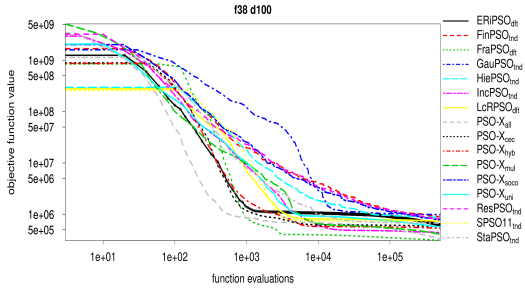
(f_{35}) Hybrid Function 1 (N = 3) - CEC'14



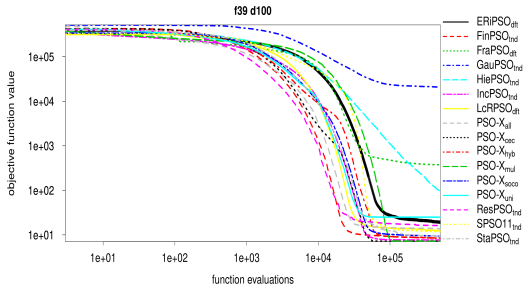
(f_{36}) Hybrid Function 2 (N = 3) - CEC'14



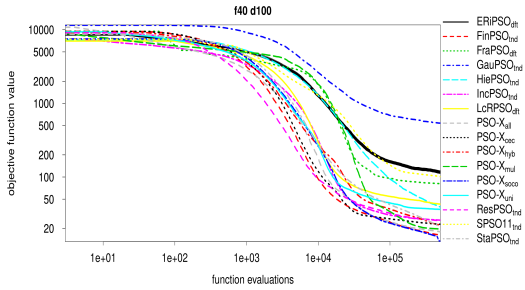
(f_{37}) Hybrid Function 3 (N = 4) - CEC'14



(f_{38}) Hybrid Function 4 (N = 4) - CEC'14



(f_{39}) Hybrid Function 5 (N = 5) - CEC'14



(f_{40}) Hybrid Function 6 (N = 5) - CEC'14

Bibliography

El-Abd, Mohammed (2011). "Opposition-based artificial bee colony algorithm". In: *Proceedings of the 13th annual conference on Genetic and evolutionary computation*, pp. 109–116.

Adenso-Dıéaz, B. and Laguna, M. (2006). "Fine-Tuning of Algorithms Using Fractional Experimental Design and Local Search". In: *Operations Research* 54.1, pp. 99–114.

Akay, Bahriye and Karaboga, Dervis (2012). "A modified artificial bee colony algorithm for real-parameter optimization". In: *Information sciences* 192, pp. 120–142.

Akbari, Reza and Ziarati, Koorush (2011). "A rank based particle swarm optimization algorithm with dynamic adaptation". In: *Journal of Computational and Applied Mathematics* 235.8, pp. 2694–2714.

Alatas, Bilal (2010). "Chaotic bee colony algorithms for global numerical optimization". In: *Expert systems with applications* 37.8, pp. 5682–5687.

Alaya, Ines, Solnon, Christine, and Ghédira, Khaled (2007). "Ant Colony Optimization for Multi-Objective Optimization Problems". In: *Proceedings of the 19th IEEE International Conference on Tools with Artificial Intelligence (ICTAI 2007)*. Vol. 1. IEEE Computer Society Press, pp. 450–457.

Alijla, Basem O, Wong, Li-Pei, Lim, Chee Peng, Khader, Ahamad Tajudin, and Al-Betar, Mohammed Azmi (2014). "A modified intelligent water drops algorithm and its application to optimization problems". In: *Expert Systems with Applications* 41.15, pp. 6555–6569.

Andréasson, Niclas, Evgrafov, Anton, and Patriksson, Michael (2020). *An Introduction to Continuous Optimization: Foundations and Fundamental Algorithms*. Courier Dover Publications.

Aranha, Claus, Camacho-Villalón, Christian Leonardo, Campelo, Felipe, Dorigo, Marco, Ruiz, Rubén, Sevaux, Marc, Sörensen, Kenneth, and Stützle, Thomas

(2022). "Metaphor-based Metaheuristics, a Call for Action: the Elephant in the Room". In: *Swarm Intelligence* 16.1, pp. 1–6.

Armas, Jesica de, Lalla-Ruiz, Eduardo, Tilahun, Surafel Luleseged, and Voß, Stefan (2022). "Similarity in metaheuristics: a gentle step towards a comparison methodology". In: *Natural Computing* 21.2, pp. 265–287.

Arumugam, M Senthil, Murthy, G Ramana, Rao, MVC, and Loo, C X (2007). "A novel effective particle swarm optimization like algorithm via extrapolation technique". In: *2007 International Conference on Intelligent and Advanced Systems*. IEEE, pp. 516–521.

Arumugam, M Senthil, Rao, Machavaram Venkata Chalapathy, and Tan, Alan WC (2009). "A novel and effective particle swarm optimization like algorithm with extrapolation technique". In: *Applied Soft Computing* 9.1, pp. 308–320.

Audet, Charles and Hare, Warren (2017). *Derivative-free and blackbox optimization*. Vol. 2. Springer.

Audet, Charles and Orban, Dominique (2006). "Finding Optimal Algorithmic Parameters Using Derivative-Free Optimization". In: *SIAM Journal on Optimization* 17.3, pp. 642–664.

Auger, Anne and Hansen, Nikolaus (2005). "A restart CMA evolution strategy with increasing population size". In: *Proceedings of the 2005 Congress on Evolutionary Computation (CEC 2005)*. IEEE Press, pp. 1769–1776.

Auger, Anne and Teytaud, Olivier (2007). "Continuous lunches are free!" In: *Proceedings of the 9th annual conference on Genetic and evolutionary computation*, pp. 916–922.

Auger, Anne and Teytaud, Olivier (2010). "Continuous lunches are free plus the design of optimal optimization algorithms". In: *Algorithmica* 57.1, pp. 121–146.

Aydın, Doğan, Yavuz, Gürcan, Özyön, Serdar, Yasar, Celal, and Stützle, Thomas (2017a). "Artificial Bee Colony Framework to Non-convex Economic Dispatch Problem with Valve Point Effects: A Case Study". In: *GECCO'17 Companion*. ACM Press, pp. 1311–1318.

Aydın, Doğan, Yavuz, Gürcan, and Stützle, Thomas (2017b). "ABC-X: A Generalized, Automatically Configurable Artificial Bee Colony Framework". In: *Swarm Intelligence* 11.1, pp. 1–38.

Aziz-Alaoui, Amine, Doerr, Carola, and Dreo, Johann (2021). "Towards large scale automated algorithm design by integrating modular benchmarking frameworks". In: *GECCO'21 Companion*. ACM Press, pp. 1365–1374.

Bäck, Thomas, Fogel, David B., and Michalewicz, Zbigniew (1997). *Handbook of evolutionary computation*. IOP Publishing.

Bäck, Thomas, Hoffmeister, Frank, and Schwefel, Hans-Paul (1991). "A survey of evolution strategies". In: *Proceedings of the fourth international conference on genetic algorithms*. Morgan Kaufmann, pp. 2–9.

Bäck, Thomas and Schwefel, Hans-Paul (1993). "An overview of evolutionary algorithms for parameter optimization". In: *Evolutionary computation* 1.1, pp. 1–23.

Baker, Monya (2016). "Reproducibility crisis". In: *Nature* 533.26, pp. 353–66.

Bartz-Beielstein, Thomas, Chiarandini, Marco, Paquete, Luís, and Preuss (Eds.), Mike (2010). *Experimental Methods for the Analysis of Optimization Algorithms*. Springer.

Bartz-Beielstein, Thomas, Doerr, Carola, Berg, Daan van den, Bossek, Jakob, Chandrasekaran, Sowmya, Eftimov, Tome, Fischbach, Andreas, Kerschke, Pascal, La Cava, William, Lopez-Ibanez, Manuel, et al. (2020). "Benchmarking in optimization: Best practice and open issues". In: *arXiv preprint arXiv:2007.03488*.

Beham, Andreas, Wagner, Stefan, and Affenzeller, Michael (2018). "Algorithm selection on generalized quadratic assignment problem landscapes". In: *Proceedings of the Genetic and Evolutionary Computation Conference, GECCO 2018*. ACM Press, pp. 253–260.

Bergh, Frans Van den and Engelbrecht, Andries Petrus (2002). "A new locally convergent particle swarm optimiser". In: *Proceedings of the 2002 IEEE international conference on systems, man and cybernetics - SMC*. Vol. 3. IEEE Press, 6–pp.

Bergh, Frans Van den and Engelbrecht, Andries Petrus (2004). "A cooperative approach to particle swarm optimization". In: *IEEE Transactions on Evolutionary Computation* 8.3, pp. 225–239.

Bezerra, Leonardo César Teonácio, López-Ibáñez, Manuel, and Stützle, Thomas (2016). "Automatic Component-Wise Design of Multi-Objective Evolutionary Algorithms". In: *IEEE Transactions on Evolutionary Computation* 20.3, pp. 403–417.

Birattari, Mauro, Balaprakash, Prasanna, and Dorigo, Marco (2006). "The ACO/F-RACE algorithm for combinatorial optimization under uncertainty". In: *Metaheuristics – Progress in Complex Systems Optimization*. Vol. 39. Operations Research/Computer Science Interfaces Series. Springer, pp. 189–203.

Blackwell, Tim and Branke, Jürgen (2004). "Multi-swarm optimization in dynamic environments". In: *Workshops on Applications of Evolutionary Computation*. Springer, pp. 489–500.

Blackwell, Tim and Branke, Jürgen (2006). "Multiswarms, exclusion, and anti-convergence in dynamic environments". In: *IEEE Transactions on Evolutionary Computation* 10.4, pp. 459–472.

Blackwell, Tim M and Bentley, Peter J (2002). "Dynamic search with charged swarms". In: *Proceedings of the 4th annual conference on genetic and evolutionary computation*, pp. 19–26.

Blum, Christian (2005). "Beam-ACO—Hybridizing Ant Colony Optimization with Beam Search: An Application to Open Shop Scheduling". In: *Computers & Operations Research* 32.6, pp. 1565–1591.

Blum, Christian and Dorigo, Marco (2004). "The hyper-cube framework for ant colony optimization". In: *IEEE Transactions on Systems, Man, and Cybernetics – Part B* 34.2, pp. 1161–1172.

Blum, Christian and Merkle, Daniel (2008). *Swarm Intelligence–Introduction and Applications*. Natural Computing Series. Springer Verlag, Berlin, Germany.

Blum, Christian and Roli, Andrea (2008). "Hybrid metaheuristics: an introduction". In: *Hybrid Metaheuristics: An emergent approach for optimization*. Vol. 114. Studies in Computational Intelligence. Springer, pp. 1–30.

Boks, Rick, Wang, Hao, and Bäck, Thomas (2020). "A modular hybridization of particle swarm optimization and differential evolution". In: *GECCO'20 Companion*. ACM Press, pp. 1418–1425.

Bonabeau, E., Dorigo, M., and Theraulaz, G. (1999). *Swarm Intelligence: From Natural to Artificial Systems*. Oxford University Press.

Bonyadi, Mohammad Reza and Michalewicz, Zbigniew (2014). "A locally convergent rotationally invariant particle swarm optimization algorithm". In: *Swarm Intelligence* 8.3, pp. 159–198.

Bonyadi, Mohammad Reza and Michalewicz, Zbigniew (2017). *Particle swarm optimization for single objective continuous space problems: a review*.

Bonyadi, Mohammad Reza, Michalewicz, Zbigniew, and Li, Xiang (2014). "An analysis of the velocity updating rule of the particle swarm optimization algorithm". In: *Journal of Heuristics* 20.4, pp. 417–452.

Booyavi, Zahra, Teymourian, Ehsan, Komaki, GM, and Sheikh, Shaya (2014). "An improved optimization method based on the intelligent water drops algorithm for the vehicle routing problem". In: *Proceedings of 2014 IEEE*

Symposium on Computational Intelligence in Production and Logistics Systems (CIPLS). IEEE Press, pp. 59–66.

Braik, Malik, Ryalat, Mohammad Hashem, and Al-Zoubi, Hussein (2022). "A novel meta-heuristic algorithm for solving numerical optimization problems: Ali Baba and the forty thieves". In: *Neural Computing and Applications* 34.1, pp. 409–455.

Bullnheimer, Bernd, Hartl, Richard F., and Strauss, Christine (1999a). "A new rank-based version of the Ant System: A computational study". In: *Central European Journal for Operations Research and Economics* 7.1, pp. 25–38.

Bullnheimer, Bernd, Hartl, Richard F., and Strauss, Christine (1999b). "An Improved Ant System Algorithm for the Vehicle Routing Problem". In: *Annals of Operations Research* 89, pp. 319–328.

Burke, Edmund K., Gendreau, Michel, Hyde, Matthew R., Kendall, Graham, Ochoa, Gabriela, Özcan, Ender, and Qu, Rong (2013). "Hyper-heuristics: A Survey of the State of the Art". In: *Journal of the Operational Research Society* 64.12, pp. 1695–1724.

Cahon, Sebastien, Melab, Nordine, and Talbi, E-G. (2004). "ParadisEO: A Framework for the Reusable Design of Parallel and Distributed Metaheuristics". In: *Journal of Heuristics* 10.3, pp. 357–380.

Camacho-Villalón, Christian Leonardo, Dorigo, Marco, and Stützle, Thomas (2018). "Why the Intelligent Water Drops Cannot Be Considered as a Novel Algorithm". In: *Swarm Intelligence, 11th International Conference, ANTS 2018*. Vol. 11172. Lecture Notes in Computer Science. Springer, pp. 302–314.

Camacho-Villalón, Christian Leonardo, Dorigo, Marco, and Stützle, Thomas (2019). "The intelligent water drops algorithm: why it cannot be considered a novel algorithm". In: *Swarm Intelligence* 13.3–4, pp. 173–192.

Camacho-Villalón, Christian Leonardo, Dorigo, Marco, and Stützle, Thomas (2021). *PSO-X: A Component-Based Framework for the Automatic Design of Particle Swarm Optimization Algorithms: Supplementary material*. http://iridia.ulb.ac.be/supp/IridiaSupp2021-001/.

Camacho-Villalón, Christian Leonardo, Dorigo, Marco, and Stützle, Thomas (2022a). "An analysis of why cuckoo search does not bring any novel ideas to optimization". In: *Computers & Operations Research* 142, p. 105747.

Camacho-Villalón, Christian Leonardo, Dorigo, Marco, and Stützle, Thomas (2022b). "PSO-X: A Component-Based Framework for the Automatic Design of Particle Swarm Optimization Algorithms". In: *IEEE Transactions on Evolutionary Computation* 26.3, pp. 402–416.

Camacho-Villalón, Christian Leonardo, Dorigo, Marco, and Stützle, Thomas (2023). "Exposing the grey wolf, moth-flame, whale, firefly, bat, and antlion algorithms: six misleading optimization techniques inspired by bestial metaphors". In: *International Transactions in Operational Research* 30.6, pp. 2945–2971.

Camacho-Villalón, Christian Leonardo, Stützle, Thomas, and Dorigo, Marco (2020). "Grey Wolf, Firefly and Bat Algorithms: Three Widespread Algorithms that Do Not Contain Any Novelty". In: *Swarm Intelligence, 12th International Conference, ANTS 2020*. Vol. 12421. Lecture Notes in Computer Science. Springer, pp. 121–133.

Campelo, Felipe and Aranha, Claus (2021a). *Evolutionary Computation Bestiary*. https://github.com/fcampelo/EC-Bestiary. Version visited last on 26 March 2021.

Campelo, Felipe and Aranha, Claus (2021b). "Sharks, Zombies and Volleyball: Lessons from the Evolutionary Computation Bestiary". In: *LIFELIKE Computing Systems Workshop 2021*.

Campelo, Felipe and Takahashi, Fernanda (2019). "Sample size estimation for power and accuracy in the experimental comparison of algorithms". In: *Journal of Heuristics* 25.2, pp. 305–338.

Černý, Vladimıér (1985). "A Thermodynamical Approach to the Traveling Salesman Problem: An Efficient Simulation Algorithm". In: *Journal of Optimization Theory and Applications* 45.1, pp. 41–51.

Clerc, Maurice (2010). *Particle swarm optimization*. Vol. 93. John Wiley & Sons.

Clerc, Maurice (2011). *Standard Particle Swarm Optimisation from 2006 to 2011*. open archive HAL hal-00764996. HAL.

Clerc, Maurice and Kennedy, James (2002). "The particle swarm-explosion, stability, and convergence in a multidimensional complex space". In: *IEEE Transactions on Evolutionary Computation* 6.1, pp. 58–73.

Codling, Edward A, Plank, Michael J, and Benhamou, Simon (2008). "Random walk models in biology". In: *Journal of the Royal society interface* 5.25, pp. 813–834.

Colorni, Alberto, Dorigo, Marco, and Maniezzo, Vittorio (1992). "Distributed Optimization by Ant Colonies". In: *Proceedings of the First European Conference on Artificial Life*. MIT Press, Cambridge, MA, pp. 134–142.

Cordón, Oscar, Viana, Iñaki Fernández de, Herrera, Francisco, and Moreno, Llanos (2000). "A New ACO Model Integrating Evolutionary Computation Concepts: The Best-Worst Ant System". In: *Abstract proceedings of ANTS 2000*

– *From Ant Colonies to Artificial Ants: Second International Workshop on Ant Algorithms*. IRIDIA, Université Libre de Bruxelles, Belgium, pp. 22–29.

Corne, David, Dorigo, Marco, Glover, Fred, Dasgupta, Dipankar, Moscato, Pablo, Poli, Riccardo, and Price, Kenneth V (1999). *New ideas in optimization*. McGraw Hill.

Crandell, KE, Howe, RO, and Falkingham, PL (2019). "Repeated evolution of drag reduction at the air–water interface in diving kingfishers". In: *Journal of the Royal Society Interface* 16.154, p. 20190125.

Cruz-Duarte, Jorge M, Ortiz-Bayliss, José C, Amaya, Iván, Shi, Yong, Terashima-Marıén, Hugo, and Pillay, Nelishia (2020). "Towards a generalised metaheuristic model for continuous optimisation problems". In: *Mathematics* 8.11, p. 2046.

Dantzig, George B and Ramser, John H (1959). "The truck dispatching problem". In: *Management science* 6.1, pp. 80–91.

Deb, Kalyanmoy (2012). *Optimization for engineering design: Algorithms and examples*. PHI Learning Pvt. Ltd.

Deb, Kalyanmoy, Pratap, A., Agarwal, S., and Meyarivan, T. (2002). "A fast and elitist multi-objective genetic algorithm: NSGA-II". In: *IEEE Transactions on Evolutionary Computation* 6.2, pp. 182–197.

Deb, Kalyanmoy, Thiele, Lothar, Laumanns, Marco, and Zitzler, Eckart (2005). "Scalable Test Problems for Evolutionary Multiobjective Optimization". In: *Evolutionary Multiobjective Optimization*. Advanced Information and Knowledge Processing. Springer, London, UK, pp. 105–145.

Deneubourg, Jean-Louis, Aron, Serge, Goss, Simon, and Pasteels, Jacques M. (1990). "The Self-Organizing Exploratory Pattern of the Argentine Ant". In: *Journal of Insect Behavior* 3.2, pp. 159–168.

Derrac, Joaquıén, Garcıéa, Salvador, Molina, Daniel, and Herrera, Francisco (2011). "A practical tutorial on the use of nonparametric statistical tests as a methodology for comparing evolutionary and swarm intelligence algorithms". In: *Swarm and Evolutionary Computation* 1.1, pp. 3–18.

Di Caro, Gianni A. and Dorigo, Marco (1998). "AntNet: Distributed Stigmergetic Control for Communications Networks". In: *Journal of Artificial Intelligence Research* 9, pp. 317–365.

Doblas, Daniel, Nebro, Antonio J, López-Ibáñez, Manuel, Garcıéa-Nieto, José, and Coello Coello, Carlos A (2022). "Automatic Design of Multi-objective Particle Swarm Optimizers". In: *Swarm Intelligence, 13th International Confer-*

ence, ANTS 2022. Vol. 13491. Lecture Notes in Computer Science. Springer, pp. 28–40.

Doerr, Carola, Wang, Hao, Ye, Furong, Van Rijn, Sander, and Bäck, Thomas (2018). "IOHprofiler: A benchmarking and profiling tool for iterative optimization heuristics". In: *arXiv preprint arXiv:1810.05281*.

Dorigo, Marco and Gambardella, Luca M. (1997a). "Ant Colony System: A Cooperative Learning Approach to the Traveling Salesman Problem". In: *IEEE Transactions on Evolutionary Computation* 1.1, pp. 53–66.

Dorigo, M. (1992a). "Optimization, Learning and Natural Algorithms (in Italian)". PhD thesis. Dipartimento di Elettronica, Politecnico di Milano.

Dorigo, M. (2001). "Ant Algorithms Solve Difficult Optimization Problems". In: *Advances in Artificial Life: 6th European Conference – ECAL 2001*. Vol. 2159. Lecture Notes in Artificial Intelligence. Springer, pp. 11–22.

Dorigo, M. and Birattari, M. (2007). "Swarm Intelligence". In: *Scholarpedia* 2.9, p. 1462.

Dorigo, M. and Stützle, T. (2018). "Ant Colony Optimization: Overview and Recent Advances". In: *Handbook of Metaheuristics, 3rd Edition*. Vol. 272. International Series in Operations Research & Management Science. Springer. Chap. 10, pp. 311–351.

Dorigo, Marco (1992b). "Optimization, Learning and Natural Algorithms". In Italian. PhD thesis. Dipartimento di Elettronica, Politecnico di Milano, Italy.

Dorigo, Marco (2016). *Swarm intelligence: A few things you need to know if you want to publish in this journal*. https://www.springer.com/cda/content/document/cda_downloaddocument/Additional_submission_instructions.pdf. Version visited last on March 26, 2021.

Dorigo, Marco and Gambardella, Luca M. (1996). *Ant Colony System*. Tech. rep. IRIDIA/96-05. IRIDIA, Université Libre de Bruxelles, Belgium.

Dorigo, Marco and Gambardella, Luca M. (1997b). "Ant Colonies for the Traveling Salesman Problem". In: *BioSystems* 43.2, pp. 73–81.

Dorigo, Marco, Maniezzo, Vittorio, and Colorni, Alberto (1991a). *Positive Feedback as a Search Strategy*. Tech. rep. 91-016. Dipartimento di Elettronica, Politecnico di Milano, Italy.

Dorigo, Marco, Maniezzo, Vittorio, and Colorni, Alberto (1991b). *The Ant System: An autocatalytic optimizing process*. Tech. rep. 91-016 Revised. Dipartimento di Elettronica, Politecnico di Milano, Italy.

Dorigo, Marco, Maniezzo, Vittorio, and Colorni, Alberto (1996). "Ant System: Optimization by a Colony of Cooperating Agents". In: *IEEE Transactions on Systems, Man, and Cybernetics – Part B* 26.1, pp. 29–41.

Dorigo, Marco and Stützle, Thomas (2004). *Ant Colony Optimization*. MIT Press.

Dréo, Johann, Liefooghe, Arnaud, Verel, Sébastien, Schoenauer, Marc, Merelo, Juan J., Quemy, Alexandre, Bouvier, Benjamin, and Gmys, Jan (2021). "Paradiseo: From a Modular Framework for Evolutionary Computation to the Automated Design of Metaheuristics: 22 Years of Paradiseo". In: *GECCO'21 Companion*. ACM Press, pp. 1522–1530.

Duan, Haibin, Liu, Senqi, and Wu, Jiang (2008). "Air robot path planning based on intelligent water drops optimization". In: *Proceedings of the International Joint Conference on Neural Networks (IJCNN 2008 - Hong Kong)*. IEEE, pp. 1397–1401.

Duan, Haibin, Liu, Senqi, and Wu, Jiang (2009). "Novel intelligent water drops optimization approach to single UCAV smooth trajectory planning". In: *Aerospace science and technology* 13.8, pp. 442–449.

Durillo, J.J., Nebro, A.J., and Alba, E. (2010). "The jMetal Framework for Multi-Objective Optimization: Design and Architecture". In: *Proceedings of the 2010 Congress on Evolutionary Computation (CEC 2010)*. IEEE Press, pp. 4138–4325.

Eberhart, Russell and Kennedy, James (1995). "A New Optimizer Using Particle Swarm Theory". In: *Proceedings of the Sixth International Symposium on Micro Machine and Human Science*, pp. 39–43.

Eberhart, Russell and Shi, Yuhui (2000). "Comparing inertia weights and constriction factors in particle swarm optimization". In: *Proceedings of the 2000 Congress on Evolutionary Computation (CEC'00)*. IEEE Press, pp. 84–88.

Edwards, Marc A and Roy, Siddhartha (2017). "Academic research in the 21st century: Maintaining scientific integrity in a climate of perverse incentives and hypercompetition". In: *Environmental engineering science* 34.1, pp. 51–61.

Eftimov, Tome, Korošec, Peter, and Seljak, Barbara Koroušić (2017). "A novel approach to statistical comparison of meta-heuristic stochastic optimization algorithms using deep statistics". In: *Information Sciences* 417, pp. 186–215.

Eftimov, Tome, Petelin, Gašper, and Korošec, Peter (2020). "DSCTool: A web-service-based framework for statistical comparison of stochastic optimization algorithms". In: *Applied Soft Computing* 87, p. 105977.

Fawcett, Chris and Hoos, Holger H. (2016). "Analysing Differences Between Algorithm Configurations through Ablation". In: *Journal of Heuristics* 22.4, pp. 431–458.

Fogel, David B., Owens, Alvin J., and Walsh, Michael J. (1966). *Artificial Intelligence Through Simulated Evolution*. John Wiley & Sons.

Fong, Simon, Wang, Xi, Xu, Qiwen, Wong, Raymond, Fiaidhi, Jinan, and Mohammed, Sabah (2016). "Recent advances in metaheuristic algorithms: Does the Makara dragon exist?" In: *The Journal of Supercomputing* 72.10, pp. 3764–3786.

Fortin, Félix-Antoine, De Rainville, François-Michel, Gardner, Marc-André, Parizeau, Marc, and Gagné, Christian (2012). "DEAP: Evolutionary Algorithms Made Easy". In: *Journal of Machine Learning Research* 13, pp. 2171–2175.

Franzin, Alberto and Stützle, Thomas (2019). "Revisiting simulated annealing: A component-based analysis". In: *Computers & Operations Research* 104, pp. 191–206.

Gambardella, Luca M. and Dorigo, Marco (1995). "Ant-Q: A Reinforcement Learning Approach to the Traveling Salesman Problem". In: *Proceedings of the Twelfth International Conference on Machine Learning (ML-95)*. Morgan Kaufmann Publishers, Palo Alto, CA, pp. 252–260.

Gambardella, Luca M. and Dorigo, Marco (1996). "Solving Symmetric and Asymmetric TSPs by Ant Colonies". In: *Proceedings of the 1996 IEEE International Conference on Evolutionary Computation (ICEC'96)*. IEEE Press, pp. 622–627.

Gambardella, Luca M., Taillard, Éric D., and Dorigo, Marco (1999). "Ant colonies for the quadratic assignment problem". In: *Journal of the Operational Research Society* 50.2, pp. 167–176.

Gambella, Claudio, Ghaddar, Bissan, and Naoum-Sawaya, Joe (2021). "Optimization problems for machine learning: A survey". In: *European Journal of Operational Research* 290.3, pp. 807–828.

Gao, Wei-feng, Liu, San-yang, and Huang, Ling-ling (2014). "Enhancing artificial bee colony algorithm using more information-based search equations". In: *Information Sciences* 270, pp. 112–133.

García-Martínez, Carlos, Gutiérrez, Pablo D, Molina, Daniel, Lozano, Manuel, and Herrera, Francisco (2017). "Since CEC 2005 competition on real-parameter optimisation: a decade of research, progress and comparative analysis's weakness". In: *Soft Computing* 21.19, pp. 5573–5583.

García-Nieto, José and Alba, Enrique (2011). "Restart particle swarm optimization with velocity modulation: a scalability test". In: *Soft Computing* 15.11, pp. 2221–2232.

Garey, M. R. and Johnson, David S. (1979). *Computers and Intractability: A Guide to the Theory of NP-Completeness*. Freeman & Co, San Francisco, CA.

Handbook of Metaheuristics (2019). Vol. 272. International Series in Operations Research & Management Science. Springer.

Glover, Fred (1986). "Future Paths for Integer Programming and Links to Artificial Intelligence". In: *Computers & Operations Research* 13.5, pp. 533–549.

Glover, Fred (1989). "Tabu Search – Part I". In: *INFORMS Journal on Computing* 1.3, pp. 190–206.

Glover, Fred (1990). "Tabu Search – Part II". In: *INFORMS Journal on Computing* 2.1, pp. 4–32.

Goldberg, David Edward (1989). *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison-Wesley.

Goss, Simon, Aron, Serge, Deneubourg, Jean-Louis, and Pasteels, Jacques Marie (1989). "Self-organized shortcuts in the Argentine ant". In: *Naturwissenschaften* 76.12, pp. 579–581.

Grefenstette, John (2000). "Rank-based selection". In: *Evolutionary computation* 1, pp. 187–194.

Guntsch, Michael and Middendorf, Martin (2002). "A Population Based Approach for ACO". In: *Applications of Evolutionary Computing, Proceedings of EvoWorkshops 2002*. Vol. 2279. Lecture Notes in Computer Science. Springer, Heidelberg, Germany, pp. 71–80.

Hansen, Michael Pilegaard (1997). "Tabu search for multiobjective optimization: MOTS". In: *Proceedings of the 13th International Conference on Multiple Criteria Decision Making (MCDM'97)*. Springer Verlag, pp. 574–586.

Hansen, Nikolaus, Arnold, Dirk V, and Auger, Anne (2015). "Evolution strategies". In: *Springer handbook of computational intelligence*. Springer, pp. 871–898.

Hansen, Nikolaus, Auger, Anne, Ros, Raymond, Mersmann, Olaf, Tušar, Tea, and Brockhoff, Dimo (2021). "COCO: A platform for comparing continuous optimizers in a black-box setting". In: *Optimization Methods and Software* 36.1, pp. 114–144.

Hansen, Nikolaus and Ostermeier, A. (2001). "Completely derandomized self-adaptation in evolution strategies". In: *Evolutionary Computation* 9.2, pp. 159–195.

Hansen, Nikolaus, Ros, Raymond, Mauny, Nikolas, Schoenauer, Marc, and Auger, Anne (2011). "Impacts of invariance in search: When CMA-ES and

PSO face ill-conditioned and non-separable problems". In: *Applied Soft Computing* 11.8, pp. 5755–5769.

Harrison, Kyle Robert, Engelbrecht, Andries Petrus, and Ombuki-Berman, Beatrice M. (2016). "Inertia weight control strategies for particle swarm optimization". In: *Swarm Intelligence* 10.4, pp. 267–305.

Harzing, Anne-Wil (2010). *The publish or perish book*. Tarma Software Research Pty Limited Melbourne.

Heppner, Frank and Grenander, Ulf (1990). "A stochastic nonlinear model for coordinated bird flocks". In: *The ubiquity of chaos*.

Herrera, Francisco, Lozano, Manuel, and Molina, D. (2010). *Test suite for the special issue of Soft Computing on scalability of evolutionary algorithms and other metaheuristics for large scale continuous optimization problems*. http://sci2s.ugr.es/eamhco/.

Holland, John Henry (1975). *Adaptation in Natural and Artificial Systems*. University of Michigan Press.

Hooker, Giles (2012). "Generalized functional ANOVA diagnostics for high-dimensional functions of dependent variables". In: *Journal of Computational and Graphical Statistics* 16.3, pp. 709–732.

Hooker, J. N. (1994). "Needed: An Empirical Science of Algorithms". In: *Operations Research* 42.2, pp. 201–212.

Hooker, J. N. (1996). "Testing Heuristics: We Have It All Wrong". In: *Journal of Heuristics* 1.1, pp. 33–42.

Hoos, Holger H. and Stützle, Thomas (2004). *Stochastic Local Search: Foundations and Applications*. Elsevier, Amsterdam, The Netherlands.

Hsieh, Sheng-Ta, Sun, Tsung-Ying, Liu, Chan-Cheng, and Tsai, Shang-Jeng (2008). "Efficient population utilization strategy for particle swarm optimizer". In: *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)* 39.2, pp. 444–456.

Huband, S., Hingston, P., Barone, L., and While, L. (2006). "A review of multiobjective test problems and a scalable test problem toolkit". In: *IEEE Transactions on Evolutionary Computation* 10.5, pp. 477–506.

Hutter, Frank, Hoos, Holger H., and Leyton-Brown, Kevin (2011). "Sequential Model-Based Optimization for General Algorithm Configuration". In: *Learning and Intelligent Optimization, 5th International Conference, LION 5*. Vol. 6683. Lecture Notes in Computer Science. Springer, Heidelberg, Germany, pp. 507–523.

Hutter, Frank, Hoos, Holger H., and Leyton-Brown, Kevin (2013). "Identifying key algorithm parameters and instance features using forward selection". In: *Learning and Intelligent Optimization, 7th International Conference, LION 7*. Vol. 7997. Lecture Notes in Computer Science. Springer, Heidelberg, Germany, pp. 364–381.

Hutter, Frank, Hoos, Holger H., and Leyton-Brown, Kevin (2014). "An Efficient Approach for Assessing Hyperparameter Importance". In: *Proceedings of the 31th International Conference on Machine Learning, ICML 2014, Beijing, China, 21-26 June 2014*. Vol. 32, pp. 754–762.

Hutter, Frank, Hoos, Holger H., Leyton-Brown, Kevin, and Stützle, Thomas (2009). "ParamILS: An Automatic Algorithm Configuration Framework". In: *Journal of Artificial Intelligence Research* 36, pp. 267–306.

Ioannidis, John PA (2005). "Why most published research findings are false". In: *PLoS medicine* 2.8, e124.

Ioannidis, John PA and Thombs, Brett D (2019). "A user's guide to inflated and manipulated impact factors". In: *European journal of clinical investigation* 49.9, e13151.

Iredi, S., Merkle, D., and Middendorf, Martin (2001). "Bi-Criterion Optimization with Multi Colony Ant Algorithms". In: *Evolutionary Multi-criterion Optimization, EMO 2001*. Vol. 1993. Lecture Notes in Computer Science. Springer, Heidelberg, Germany, pp. 359–372.

Iwamatsu, Masao (2002). "Generalized evolutionary programming with Levy-type mutation". In: *Computer Physics Communications* 147.1-2, pp. 729–732.

Janson, Stefan and Middendorf, Martin (2005). "A hierarchical particle swarm optimizer and its adaptive variant". In: *IEEE Transactions on Systems, Man, and Cybernetics – Part B* 35.6, pp. 1272–1282.

Jordan, Johannes, Helwig, Sabine, and Wanka, Rolf (2008). "Social interaction in particle swarm optimization, the ranked FIPS, and adaptive multi-swarms". In: *Proceedings of the Genetic and Evolutionary Computation Conference, GECCO 2008*. ACM Press, pp. 49–56.

Kappler, Cornelia (1996). "Are evolutionary algorithms improved by large mutations?" In: *International Conference on Parallel Problem Solving from Nature*. Springer, pp. 346–355.

Karaboga, Dervis et al. (2005). *An idea based on honey bee swarm for numerical optimization*. Tech. rep. Technical report-tr06, Erciyes university, engineering faculty, computer ...

Karaboga, Dervis and Akay, Bahriye (2009). "A Survey: Algorithms Simulating Bee Swarm Intelligence". In: *Artificial Intelligence Review* 31.1–4, pp. 61–85.

Karaboga, Dervis and Basturk, Bahriye (2007). "A powerful and efficient algorithm for numerical function optimization: artificial bee colony (ABC) algorithm". In: *Journal of Global Optimization* 39.3, pp. 459–471.

Karimi-Mamaghan, Maryam, Mohammadi, Mehrdad, Meyer, Patrick, Karimi-Mamaghan, Amir Mohammad, and Talbi, El-Ghazali (2022). "Machine learning at the service of meta-heuristics for solving combinatorial optimization problems: A state-of-the-art". In: *European Journal of Operational Research* 296.2, pp. 393–422.

Keijzer, Maarten, Merelo, Juan J, Romero, Gustavo, and Schoenauer, Marc (2002). "Evolving objects: A general purpose evolutionary computation library". In: *Artificial Evolution: 5th International Conference, Evolution Artificielle, EA 2001 Le Creusot, France, October 29–31, 2001 Selected Papers 5*. Springer, pp. 231–242.

Kennedy, J., Eberhart, R. C., and Shi, Y. (2001). *Swarm Intelligence*. Morgan Kaufmann Publishers, San Francisco, CA.

Kennedy, James (1999). "Small worlds and mega-minds: effects of neighborhood topology on particle swarm performance". In: *Proceedings of the 1999 congress on evolutionary computation-CEC99 (Cat. No. 99TH8406)*. Vol. 3. IEEE, pp. 1931–1938.

Kennedy, James (2003). "Bare bones particle swarms". In: *Proceedings of the 2003 IEEE Swarm Intelligence Symposium. SIS'03 (Cat. No. 03EX706)*. IEEE, pp. 80–87.

Kennedy, James and Eberhart, Russell (1995). "Particle swarm optimization". In: *Proceedings of ICNN'95-International Conference on Neural Networks*. Vol. 4. IEEE, pp. 1942–1948.

Kennedy, James and Mendes, Rui (2002). "Population structure and particle swarm performance". In: *Proceedings of the 2002 Congress on Evolutionary Computation (CEC'02)*. IEEE Press, pp. 1671–1676.

KhudaBukhsh, A. R., Xu, Lin, Hoos, Holger H., and Leyton-Brown, Kevin (2009). "SATenstein: Automatically Building Local Search SAT Solvers from Components". In: *IJCAI 2009, Proceedings of the 21st International Joint Conference on Artificial Intelligence, Pasadena, California, USA, July 11-17, 2009*. AAAI Press, Menlo Park, CA, pp. 517–524.

KhudaBukhsh, A. R., Xu, Lin, Hoos, Holger H., and Leyton-Brown, Kevin (2016). "SATenstein: Automatically Building Local Search SAT Solvers from Components". In: *Artificial Intelligence* 232, pp. 20–42.

Kirkpatrick, Scott (1984). "Optimization by Simulated Annealing: Quantitative Studies". In: *Journal of Statistical Physics* 34.5-6, pp. 975–986.

Kirkpatrick, Scott, Gelatt, C. D., and Vecchi, M. P. (1983). "Optimization by Simulated Annealing". In: *Science* 220, pp. 671–680.

Knowles, Joshua D., Thiele, Lothar, and Zitzler, Eckart (2006). *A tutorial on the performance assessment of stochastic multiobjective optimizers*. TIK-Report 214. Revised version. Computer Engineering and Networks Laboratory (TIK) – Swiss Federal Institute of Technology (ETH), Zürich, Switzerland.

Koza, J. (1992). *Genetic Programming: On the Programming of Computers By the Means of Natural Selection*. MIT Press.

Kudela, Jakub (2022). "A critical problem in benchmarking and analysis of evolutionary computation methods". In: *Nature Machine Intelligence* 4.12, pp. 1238–1245.

Kudela, Jakub (2023). "The Evolutionary Computation Methods No One Should Use". In: *arXiv preprint arXiv:2301.01984*.

Kukkonen, S. and Lampinen, J. (2005). "GDE3: the third evolution step of generalized differential evolution". In: *Proceedings of the 2005 Congress on Evolutionary Computation (CEC 2005)*. IEEE Press, pp. 443–450.

Lee, Chang-Yong and Yao, Xin (2004). "Evolutionary programming using mutations based on the Lévy probability distribution". In: *IEEE Transactions on Evolutionary Computation* 8.1, pp. 1–13.

Leguizamón, Guillermo and Coello, Carlos A Coello (2010). "An Alternative ACO _R Algorithm for Continuous Optimization Problems". In: *Swarm Intelligence: 7th International Conference, ANTS 2010, Brussels, Belgium, September 8-10, 2010. Proceedings 7*. Springer, pp. 48–59.

Lehre, Per Kristian and Witt, Carsten (2013). "Finite first hitting time versus stochastic convergence in particle swarm optimisation". In: *Advances in metaheuristics*. Springer, pp. 1–20.

Li, Xianneng and Yang, Guangfei (2016). "Artificial bee colony algorithm with memory". In: *Applied Soft Computing* 41, pp. 362–372.

Li, Xiaodong and Yao, Xin (2011). "Cooperatively coevolving particle swarms for large scale optimization". In: *IEEE Transactions on Evolutionary Computation* 16.2, pp. 210–224.

Liang, J.J., Qu, B.Y., and Suganthan, Ponnuthurai N. (2005). "Problem definitions and evaluation criteria for the CEC 2014 special session and competition on single objective real-parameter numerical optimization". In.

Liao, T., Stützle, T., Montes de Oca, M. A., and Dorigo, M. (2014a). "A Unified Ant Colony Optimization Algorithm for Continuous Optimization". In: *European Journal of Operational Research* 234.3, pp. 597–609.

Liao, Tianjun, Montes de Oca, Marco A., Aydın, Doğan, Stützle, Thomas, and Dorigo, Marco (2011). "An Incremental Ant Colony Algorithm with Local Search for Continuous Optimization". In: *Proceedings of the Genetic and Evolutionary Computation Conference, GECCO 2011*. ACM Press, pp. 125–132.

Liao, Tianjun, Socha, K., Montes de Oca, Marco A., Stützle, Thomas, and Dorigo, Marco (2014b). "Ant Colony Optimization for Mixed-Variable Optimization Problems". In: *IEEE Transactions on Evolutionary Computation* 18.4, pp. 503–518.

Lones, Michael A (2014). "Metaheuristics in nature-inspired algorithms". In: *Proceedings of the Genetic and Evolutionary Computation Conference, GECCO 2014*. ACM Press, pp. 1419–1422.

Lones, Michael A (2020). "Mitigating metaphors: A comprehensible guide to recent nature-inspired algorithms". In: *SN Computer Science* 1.1, pp. 1–12.

López-Ibáñez, Manuel, Branke, Juergen, and Paquete, Luıés (2021). "Reproducibility in evolutionary computation". In: *ACM Transactions on Evolutionary Learning and Optimization* 1.4, pp. 1–21.

López-Ibáñez, Manuel, Dubois-Lacoste, Jérémie, Pérez Cáceres, Leslie, Stützle, Thomas, and Birattari, Mauro (2016). "The irace package: Iterated Racing for Automatic Algorithm Configuration". In: *Operations Research Perspectives* 3, pp. 43–58.

López-Ibáñez, Manuel and Stützle, Thomas (2012). "The Automatic Design of Multi-Objective Ant Colony Optimization Algorithms". In: *IEEE Transactions on Evolutionary Computation* 16.6, pp. 861–875.

López-Ibáñez, Manuel, Stützle, Thomas, and Dorigo, Marco (2017). "Ant Colony Optimization: A Component-Wise Overview". In: *Handbook of Heuristics*. Springer International Publishing, pp. 1–37.

Lu, Peng, Zhou, Jianzhong, Zhang, Huifeng, Zhang, Rui, and Wang, Chao (2014). "Chaotic differential bee colony optimization algorithm for dynamic economic dispatch problem with valve-point effects". In: *International Journal of Electrical Power & Energy Systems* 62, pp. 130–143.

Luenberger, David G, Ye, Yinyu, et al. (2016). *Linear and Nonlinear Programming*. Vol. 2. Springer.

Ma, Zhongqiang, Wu, Guohua, Suganthan, Ponnuthurai Nagaratnam, Song, Aijuan, and Luo, Qizhang (2023). "Performance assessment and exhaustive

listing of 500+ nature-inspired metaheuristic algorithms". In: *Swarm and Evolutionary Computation* 77, p. 101248.

Maniezzo, Vittorio (1999). "Exact and Approximate Nondeterministic Tree-Search Procedures for the Quadratic Assignment Problem". In: *INFORMS Journal on Computing* 11.4, pp. 358–369.

Maniezzo, Vittorio, Boschetti, Marco Antonio, and Stützle, Thomas (2022). *Matheuristics — Algorithms and Implementations*. EURO Advanced Tutorials on Operational Research. Springer Cham.

Maniezzo, Vittorio and Carbonaro, A. (2000). "An ANTS Heuristic for the Frequency Assignment Problem". In: *Future Generation Computer Systems* 16.8, pp. 927–935.

Maniezzo, Vittorio and Colorni, Alberto (1999). "The Ant System Applied to the Quadratic Assignment Problem". In: *IEEE Transactions on Data and Knowledge Engineering* 11.5, pp. 769–778.

Maron, O. and Moore, A. W. (1997). "The Racing Algorithm: Model Selection for Lazy Learners". In: *Artificial Intelligence Research* 11.1–5, pp. 193–225.

Mascia, Franco, López-Ibáñez, Manuel, Dubois-Lacoste, Jérémie, and Stützle, Thomas (2014). "Grammar-based generation of stochastic local search heuristics through automatic algorithm configuration tools". In: *Computers & Operations Research* 51, pp. 190–199.

Melvin, Gauci, Dodd, Tony J., and Groß, Roderich (2012). "Why 'GSA: a gravitational search algorithm' is not genuinely based on the law of gravity". In: *Natural Computing* 11.4, pp. 719–720.

Mendes, Rui (2004). "Population topologies and their influence in particle swarm performance". In: *PhD Final Dissertation, Departamento de Informática Escola de Engenharia Universidade do Minho*.

Mendes, Rui, Kennedy, James, and Neves, José (2004). "The fully informed particle swarm: simpler, maybe better". In: *IEEE Transactions on Evolutionary Computation* 8.3, pp. 204–210.

Merritt, Wendy S, Letcher, Rebecca A, and Jakeman, Anthony J (2003). "A review of erosion and sediment transport models". In: *Environmental modelling & software* 18.8-9, pp. 761–799.

Metaheuristics Network. Project Summary (n.d.). http://www.metaheuristics.net/. Version visited last on March 26, 2023.

Michalewicz, Zbigniew and Schoenauer, Marc (2013). "Evolutionary Algorithms". In: *Encyclopedia of Operations Research and Management Science*. Springer Verlag, pp. 517–527.

Miranda, Péricles Barbosa and Prudêncio, Ricardo Bastos (2015). "Gefpso: A framework for pso optimization based on grammatical evolution". In: *Proceedings of the 2015 Annual Conference on Genetic and Evolutionary Computation*, pp. 1087–1094.

Mirjalili, Seyedali (2015a). "Moth-flame optimization algorithm: A novel nature-inspired heuristic paradigm". In: *Knowledge-based systems* 89, pp. 228–249.

Mirjalili, Seyedali (2015b). "The ant lion optimizer". In: *Advances in engineering software* 83, pp. 80–98.

Mirjalili, Seyedali and Lewis, Andrew (2016). "The whale optimization algorithm". In: *Advances in Engineering Software* 95, pp. 51–67.

Mirjalili, Seyedali, Mirjalili, Seyed Mohammad, and Lewis, Andrew (2014). "Grey wolf optimizer". In: *Advances in Engineering Software* 69, pp. 46–61.

Molina, Daniel, Poyatos, Javier, Ser, Javier Del, Garcıéa, Salvador, Hussain, Amir, and Herrera, Francisco (2020). "Comprehensive taxonomies of nature-and bio-inspired optimization: Inspiration versus algorithmic behavior, critical analysis recommendations". In: *Cognitive Computation* 12.5, pp. 897–939.

Montes de Oca, Marco A. (2011). "Incremental Social Learning in Swarm Intelligence Systems". PhD thesis. IRIDIA, École polytechnique, Université Libre de Bruxelles, Belgium.

Montes de Oca, Marco A., Stützle, Thomas, Birattari, Mauro, and Dorigo, Marco (2009). "Frankenstein's PSO: A Composite Particle Swarm Optimization Algorithm". In: *IEEE Transactions on Evolutionary Computation* 13.5, pp. 1120–1132.

Montes de Oca, Marco A., Stützle, Thomas, Van den Enden, Ken, and Dorigo, Marco (2010). "Incremental social learning in particle swarms". In: *IEEE Transactions on Systems, Man, and Cybernetics – Part B* 41.2, pp. 368–384.

Msallam, Mohammed M and Hamdan, Mohammad (2011). "Improved intelligent water drops algorithm using adaptive schema". In: *International Journal of Bio-Inspired Computation* 3.2, pp. 103–111.

Nannen, V. and Eiben, Agoston E. (2006). "A Method for Parameter Calibration and Relevance Estimation in Evolutionary Algorithms". In: *Proceedings of the Genetic and Evolutionary Computation Conference, GECCO 2006*. ACM Press, pp. 183–190.

Nebro, Antonio J, López-Ibáñez, Manuel, Barba-González, Cristóbal, and García-Nieto, José (2019). "Automatic configuration of NSGA-II with jMetal and irace". In: *Proceedings of the Genetic and Evolutionary Computation Conference, GECCO 2019*. ACM Press, pp. 1374–1381.

Nelder, John A and Mead, Roger (1965). "A simplex method for function minimization". In: *The computer journal* 7.4, pp. 308–313.

Niu, SH, Ong, Soh Khim, and Nee, Andrew YC (2012). "An improved intelligent water drops algorithm for achieving optimal job-shop scheduling solutions". In: *International Journal of Production Research* 50.15, pp. 4192–4205.

Nobel, Jacob de, Vermetten, Diederick, Wang, Hao, Doerr, Carola, and Bäck, Thomas (2021). "Tuning as a means of assessing the benefits of new ideas in interplay with existing algorithmic modules". In: *GECCO'21 Companion*. ACM Press, pp. 1375–1384.

Oldewage, Elre T, Engelbrecht, Andries P, and Cleghorn, Christopher W (2020). "Movement patterns of a particle swarm in high dimensional spaces". In: *Information Sciences* 512, pp. 1043–1062.

Pagnozzi, Federico and Stützle, Thomas (2019). "Automatic design of hybrid stochastic local search algorithms for permutation flowshop problems". In: *European Journal of Operational Research* 276.2, pp. 409–421.

Papadimitriou, Christos H. and Steiglitz, K. (1982). *Combinatorial Optimization – Algorithms and Complexity*. Prentice Hall, Englewood Cliffs, NJ.

Peña, Jorge (2008a). "Simple dynamic particle swarms without velocity". In: *Ant Colony Optimization and Swarm Intelligence, 6th International Conference, ANTS 2008*. Vol. 5217. Lecture Notes in Computer Science. Springer, Heidelberg, Germany, pp. 144–154.

Peña, Jorge (2008b). "Theoretical and empirical study of particle swarms with additive stochasticity and different recombination operators". In: *Proceedings of the Genetic and Evolutionary Computation Conference, GECCO 2008*. ACM Press, pp. 95–102.

Pérez Cáceres, Leslie, Bischl, Bernd, and Stützle, Thomas (2017). "Evaluating random forest models for irace". In: *GECCO'17 Companion*. ACM Press, pp. 1146–1153.

Piotrowski, Adam P., Napiorkowski, Jaroslaw J., and Rowinski, Pawel M. (2014). "How novel is the "novel" black hole optimization approach?" In: *Information Sciences* 267, pp. 191–200.

Poli, Riccardo (2009). "Mean and variance of the sampling distribution of particle swarm optimizers during stagnation". In: *IEEE Transactions on Evolutionary Computation* 13.4, pp. 712–721.

Poli, Riccardo and Broomhead, David (2007). "Exact analysis of the sampling distribution for the canonical particle swarm optimiser and its convergence

during stagnation". In: *Proceedings of the Genetic and Evolutionary Computation Conference, GECCO 2007*. ACM Press, pp. 134–141.

Poli, Riccardo, Kennedy, James, and Blackwell, Tim (2007). "Particle swarm optimization". In: *Swarm Intelligence* 1.1, pp. 33–57.

Poli, Riccardo, Langdon, William B, and Holland, Owen (2005). "Extending particle swarm optimisation via genetic programming". In: *European Conference on Genetic Programming*. Springer, pp. 291–300.

Powell, Michael James David et al. (1981). *Approximation theory and methods*. Cambridge university press.

Price, Kenneth, Storn, Rainer M., and Lampinen, Jouni A. (2005). *Differential Evolution: A Practical Approach to Global Optimization*. Springer, New York, NY.

R Development Core Team (2008). *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing.

Ramalhinho Lourenço, Helena, Martin, Olivier, and Stützle, Thomas (2002). "Iterated Local Search". In: *Handbook of Metaheuristics*. Kluwer Academic Publishers, Norwell, MA, pp. 321–353.

Ratnaweera, Asanga, Halgamuge, Saman K, and Watson, Harry C (2004). "Self-organizing hierarchical particle swarm optimizer with time-varying acceleration coefficients". In: *IEEE Transactions on Evolutionary Computation* 8.3, pp. 240–255.

Rechenberg, I. (1971). "Evolutionsstrategie: Optimierung technischer Systeme nach Prinzipien der biologischen Evolution". PhD thesis. Department of Process Engineering, Technical University of Berlin.

Rechenberg, I. (1973). *Evolutionsstrategie: Optimierung technischer Systeme nach Prinzipien der biologischen Evolution*. Frommann-Holzboog, Stuttgart, Germany.

Richer, Toby J and Blackwell, Tim (2006). "The Lévy particle swarm". In: *Proceedings of the 2006 Congress on Evolutionary Computation (CEC 2006)*. IEEE Press, pp. 808–815.

Ros, Raymond and Hansen, Nikolaus (2008). "A simple modification in CMA-ES achieving linear time and space complexity". In: *Parallel Problem Solving from Nature–PPSN X: 10th International Conference, Dortmund, Germany, September 13-17, 2008. Proceedings 10*. Springer, pp. 296–305.

Rothlauf, Franz (2011). *Design of modern heuristics: principles and application*. Natural Computing Series. Springer.

Sabar, Nasser R, Ayob, Masri, Kendall, Graham, and Qu, Rong (2013). "Grammatical evolution hyper-heuristic for combinatorial optimization problems". In: *IEEE Transactions on Evolutionary Computation* 17.6, pp. 840–861.

Sagan, Carl (1979). *Broca's Brain: Reflections on the Romance of Science*. Random House.

Schaffer, J. David (1985). "Multiple Objective Optimization with Vector Evaluated Genetic Algorithms". In: *Proceedings of the 1st International Conference on Genetic Algorithms, Pittsburgh, PA, USA, July 1985*. Lawrence Erlbaum Associates, pp. 93–100.

Schmitt, Manuel and Wanka, Rolf (2013). "Particles prefer walking along the axes: Experimental insights into the behavior of a particle swarm". In: *Proceedings of the Genetic and Evolutionary Computation Conference, GECCO 2013*. ACM Press, pp. 17–18.

Schwefel, Hans-Paul (1977). *Numerische Optimierung von Computer–Modellen mittels der Evolutionsstrategie*. Birkhäuser, Basel, Switzerland.

Schwefel, Hans-Paul (1981). *Numerical Optimization of Computer Models*. John Wiley & Sons, Inc.

Shah-Hosseini, Hamed (2007). "Problem solving by intelligent water drops". In: *Proceedings of the 2007 Congress on Evolutionary Computation (CEC 2007)*. IEEE. IEEE Press, pp. 3226–3231.

Shah-Hosseini, Hamed (2008). "Intelligent water drops algorithm: A new optimization method for solving the multiple knapsack problem". In: *International Journal of Intelligent Computing and Cybernetics* 1.2, pp. 193–212.

Shah-Hosseini, Hamed (2009). "The intelligent water drops algorithm: a nature-inspired swarm-based optimization algorithm". In: *International Journal of Bio-Inspired Computation* 1.1-2, pp. 71–79.

Sheskin, David J. (2011). *Handbook of Parametric and Nonparametric Statistical Procedures*. Chapman & Hall/CRC.

Shi, Yuhui and Eberhart, Russell (1998). "A modified particle swarm optimizer". In: *Proceedings of the 1998 IEEE International Conference on Evolutionary Computation (ICEC'98)*. IEEE Press, pp. 69–73.

Shi, Yuhui and Eberhart, Russell (1999). "Empirical study of particle swarm optimization". In: *Proceedings of the 1999 Congress on Evolutionary Computation (CEC 1999)*. IEEE Press, pp. 1945–1950.

Simon, Dan (2008). "Biogeography-based optimization". In: *IEEE Transactions on Evolutionary Computation* 12.6, pp. 702–713.

Simon, Dan, Rarick, Rick, Ergezer, Mehmet, and Du, Dawei (2011). "Analytical and numerical comparisons of biogeography-based optimization and genetic algorithms". In: *Information Sciences* 181.7, pp. 1224–1248.

Socha, K. and Dorigo, Marco (2008). "Ant Colony Optimization for Continuous Domains". In: *European Journal of Operational Research* 185.3, pp. 1155–1173.

Song, Heda, Triguero, Isaac, and Özcan, Ender (2019). "A review on the self and dual interactions between machine learning and optimisation". In: *Progress in Artificial Intelligence* 8.2, pp. 143–165.

Sörensen, Kenneth (2015). "Metaheuristics—the metaphor exposed". In: *International Transactions in Operational Research* 22.1, pp. 3–18.

Sörensen, Kenneth, Arnold, Florian, and Palhazi Cuervo, Daniel (2019). "A critical analysis of the "improved Clarke and Wright savings algorithm"". In: *International Transactions in Operational Research* 26.1, pp. 54–63.

Sörensen, Kenneth, Sevaux, Marc, and Glover, Fred (2018). "A history of metaheuristics". In: *Handbook of heuristics*. Springer, pp. 791–808.

Stegherr, Helena, Heider, Michael, and Hähner, Jörg (2020). "Classifying Metaheuristics: Towards a unified multi-level classification system". In: *Natural Computing* 21.2, pp. 1–17.

Storn, Rainer and Price, Kenneth (1997). "Differential Evolution – A Simple and Efficient Heuristic for Global Optimization over Continuous Spaces". In: *Journal of Global Optimization* 11.4, pp. 341–359.

Stützle, Thomas and Hoos, Holger H. (1996). *Improving the Ant System: A Detailed Report on the $\mathcal{MAX}$–$\mathcal{MIN}$ Ant System*. Tech. rep. AIDA–96–12. FG Intellektik, FB Informatik, TU Darmstadt, Germany.

Stützle, Thomas and Hoos, Holger H. (1997). "The $\mathcal{MAX}$–$\mathcal{MIN}$ Ant System and Local Search for the Traveling Salesman Problem". In: *Proceedings of the 1997 IEEE International Conference on Evolutionary Computation (ICEC'97)*. IEEE Press, pp. 309–314.

Stützle, Thomas and Hoos, Holger H. (2000). "$\mathcal{MAX}$–$\mathcal{MIN}$ Ant System". In: *Future Generation Computer Systems* 16.8, pp. 889–914.

Stützle, Thomas and López-Ibáñez, Manuel (2019). "Automated Design of Metaheuristic Algorithms". In: *Handbook of Metaheuristics*. Vol. 272. International Series in Operations Research & Management Science. Springer, pp. 541–579.

Stützle, Thomas, López-Ibáñez, Manuel, Pellegrini, Paola, Maur, Michael, Montes de Oca, Marco A., Birattari, Mauro, and Dorigo, Marco (2012). "Parameter Adaptation in Ant Colony Optimization". In: *Autonomous Search*. Springer, pp. 191–215.

Suganthan, Ponnuthurai N., Hansen, Nikolaus, Liang, J. J., Deb, Kalyanmoy, Chen, Y. P., Auger, A., and Tiwari, S. (2005). *Problem definitions and evaluation criteria for the CEC 2005 special session on real-parameter optimization*. Tech. rep. Nanyang Technological University, Singapore.

Swan, Jerry, Adriænsen, Steven, Barwell, Adam D, Hammond, Kevin, and White, David R (2019). "Extending the "Open-Closed Principle" to Automated Algorithm Configuration". In: *Evolutionary computation* 27.1, pp. 173–193.

Swan, Jerry, Adriaensen, Steven, Brownlee, Alexander EI, Hammond, Kevin, Johnson, Colin G, Kheiri, Ahmed, Krawiec, Faustyna, Merelo, Juan Julián, Minku, Leandro L, Özcan, Ender, et al. (2022). "Metaheuristics 'in the large'". In: *European Journal of Operational Research* 297.2, pp. 393–406.

Talbi, E-G. (2002). "A Taxonomy of Hybrid Metaheuristics". In: *Journal of Heuristics* 8.5, pp. 541–564.

Hybrid Metaheuristics (2013). Vol. 434. Studies in Computational Intelligence. Springer Verlag.

Talbi, El-Ghazali (2021). "Machine learning into metaheuristics: A survey and taxonomy". In: *ACM Computing Surveys (CSUR)* 54.6, pp. 1–32.

Teymourian, Ehsan, Kayvanfar, Vahid, Komaki, GH M, and Khodarahmi, Majtaba (2016a). "An enhanced intelligent water drops algorithm for scheduling of an agile manufacturing system". In: *International Journal of Information Technology & Decision Making* 15.02, pp. 239–266.

Teymourian, Ehsan, Kayvanfar, Vahid, Komaki, GH M, and Zandieh, Mostafa (2016b). "Enhanced intelligent water drops and cuckoo search algorithms for solving the capacitated vehicle routing problem". In: *Information Sciences* 334, pp. 354–378.

Thymianis, Marios and Tzanetos, Alexandros (2022). "Is integration of mechanisms a way to enhance a nature-inspired algorithm?" In: *Natural Computing*, pp. 1–21.

Tovey, Craig A (2002). "Tutorial on computational complexity". In: *Interfaces* 32.3, pp. 30–61.

Trelea, Ioan Cristian (2003). "The particle swarm optimization algorithm: convergence analysis and parameter selection". In: *Information Processing Letters* 85.6, pp. 317–325.

Tzanetos, Alexandros and Dounias, Georgios (2021). "Nature inspired optimization algorithms or simply variations of metaheuristics?" In: *Artificial Intelligence Review* 54.3, pp. 1841–1862.

Tzanetos, Alexandros, Fister Jr, Iztok, and Dounias, Georgios (2020). "A comprehensive database of Nature-Inspired Algorithms". In: *Data in Brief* 31, p. 105792.

Van Veldhuizen, David A. and Lamont, Gary B. (1998). "Evolutionary Computation and Convergence to a Pareto Front". In: *Genetic Programming 1998: Proceedings of the Third Annual Conference, Late Breaking Papers*. Stanford University Bookstore, pp. 221–228.

Wagner, Stefan and Affenzeller, Michael (2005). "Heuristiclab: A generic and extensible optimization environment". In: *Adaptive and Natural Computing Algorithms, Proceedings of the International Conference in Coimbra, Portugal, 2005*. Springer, pp. 538–541.

Wang, Yayun, Naleway, Steven E, and Wang, Bin (2020). "Biological and bioinspired materials: Structure leading to functional and mechanical performance". In: *Bioactive materials* 5.4, pp. 745–757.

Weyland, Dennis (2010). "A Rigorous Analysis of the Harmony Search Algorithm: How the Research Community can be misled by a "novel" Methodology". In: *International Journal of Applied Metaheuristic Computing* 12.2, pp. 50–60.

Wilke, Daniel N, Kok, Schalk, and Groenwold, Albert A (2007). "Comparison of linear and classical velocity update rules in particle swarm optimization: Notes on scale and frame invariance". In: *International Journal for Numerical Methods in Engineering* 70.8, pp. 985–1008.

Wolpert, D. H. and Macready, W. G. (1997). "No Free Lunch Theorems for Optimization". In: *IEEE Transactions on Evolutionary Computation* 1.1, pp. 67–82.

Xiang, Wan-li and An, Mei-qing (2013). "An efficient and robust artificial bee colony algorithm for numerical optimization". In: *Computers & Operations Research* 40.5, pp. 1256–1265.

Xiang, Wanli, Ma, Shoufeng, and An, Meiqing (2014). "Habcde: a hybrid evolutionary algorithm based on artificial bee colony algorithm and differential evolution". In: *Applied Mathematics and Computation* 238, pp. 370–386.

Xinchao, Zhao (2010). "A perturbed particle swarm algorithm for numerical optimization". In: *Applied Soft Computing* 10.1, pp. 119–124.

Yang, Xin-She (2009). "Firefly algorithms for multimodal optimization". In: *International symposium on stochastic algorithms*. Springer, pp. 169–178.

Yang, Xin-She (2010). "A new metaheuristic bat-inspired algorithm". In: *Nature Inspired Cooperative Strategies for Optimization (NICSO 2010)*. Vol. 284. Studies in Computational Intelligence. Springer, pp. 65–74.

Yang, Xin-She (2021). *Cuckoo Search (CS) Algorithm*. https://www.mathworks.com/matlabcentral/fileexchange/29809-cuckoo-search-cs-algorithm. MATLAB Central File Exchange. Retrieved March 12, 2021.

Yang, Xin-She and Deb, Suash (2009). "Cuckoo search via Lévy flights". In: *2009 World Congress on Nature & Biologically Inspired Computing (NaBIC)*, pp. 210–214.

Yang, Xin-She and Deb, Suash (2010). "Engineering optimisation by cuckoo search". In: *International Journal of Mathematical Modelling and Numerical Optimisation* 1.4, pp. 330–343.

Yuan, Zhi, Montes de Oca, Marco A., Stützle, Thomas, and Birattari, Mauro (2012). "Continuous Optimization Algorithms for Tuning Real and Integer Algorithm Parameters of Swarm Intelligence Algorithms". In: *Swarm Intelligence* 6.1, pp. 49–75.

Zambrano-Bigiarin, Mauricio, Clerc, Maurice, and Rojas, Rodrigo (2013). "Standard particle swarm optimisation 2011 at cec-2013: A baseline for future pso improvements". In: *Proceedings of the 2013 Congress on Evolutionary Computation (CEC 2013)*. IEEE Press, pp. 2337–2344.

Zitzler, Eckart and Künzli, Simon (2004). "Indicator-based Selection in Multiobjective Search". In: *Proceedings of PPSN-VIII, Eigth International Conference on Parallel Problem Solving from Nature*. Vol. 3242. Lecture Notes in Computer Science. Springer, Heidelberg, Germany, pp. 832–842.

Zitzler, Eckart and Thiele, Lothar (1999). "Multiobjective Evolutionary Algorithms: A Comparative Case Study and the Strength Pareto Evolutionary Algorithm". In: *IEEE Transactions on Evolutionary Computation* 3.4, pp. 257–271.

Zitzler, Eckart, Thiele, Lothar, and Deb, Kalyanmoy (2000). "Comparison of Multiobjective Evolutionary Algorithms: Empirical Results". In: *Evolutionary Computation* 8.2, pp. 173–195.

Zyl, ET van and Engelbrecht, Andries P (2016). "Group-based stochastic scaling for PSO velocities". In: *2016 IEEE Congress on Evolutionary Computation (CEC)*. IEEE, pp. 1862–1868.