UNIVERSIDADE FEDERAL DE PERNAMBUCO
CENTRO DE INFORMÁTICA
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

MARCELO GOMES PEREIRA DE LACERDA

Out-of-the-box Parameter Control for Evolutionary and Swarm-based Algorithms with Distributed Reinforcement Learning

Recife

2021

MARCELO GOMES PEREIRA DE LACERDA

Out-of-the-box Parameter Control for Evolutionary and Swarm-based Algorithms with Distributed Reinforcement Learning

Tese apresentada ao Programa de Pós-Graduação em Ciências da Computação da Universidade Federal de Pernambuco, como requisito parcial para a obtenção do título de Doutor em Ciências da Computação.

Área de Concentração: Inteligência Computacional.

Orientadora: Profa. Teresa Bernarda Ludermir, Ph.D.
Co-Orientador: Prof. Fernando Buarque de Lima Neto, Ph.D.

Recife
2021

Catalogação na fonte
Bibliotecário Cristiano Cosme S. dos Anjos, CRB4-2290

L131o Lacerda, Marcelo Gomes Pereira de
Out-of-the-box parameter control for evolutionary and swarm-based algorithms with distributed reinforcement learning / Marcelo Gomes Pereira de Lacerda. – 2021.
213 f.: il., fig., tab.

Orientadora: Teresa Bernarda Ludermir.
Tese (Doutorado) – Universidade Federal de Pernambuco. CIn, Ciência da Computação, Recife, 2021.
Inclui referências e apêndices.

1. Inteligência Computacional. 2. Inteligência de enxames. 3. Computação evolucionária. 4. Aprendizagem por reforço. I. Ludermir, Teresa Bernarda (orientadora). II. Título.

006.31 CDD (23. ed.) UFPE - CCEN 2021 – 95

Marcelo Gomes Pereira de Lacerda

"Out-of-the-box Parameter Control for Evolutionary and Swarm-based Algorithms with Distributed Reinforcement Learning"

Tese de Doutorado apresentada ao Programa de Pós-Graduação em Ciência da Computação da Universidade Federal de Pernambuco, como requisito parcial para a obtenção do título de Doutor em Ciência da Computação.

Aprovado em: 19/03/2021.

Dedicated to all health professionals and scientists that abdicated their lives to fight on the battle front of the war against COVID-19.

ACKNOWLEDGEMENTS

Firstly, I would like to thank the creator and ruler of this universe for allowing me to walk the path that led me to where I am today. In addition, I would like to thank the Brazilian government and the State of Pernambuco for having financed my entire journey in academia so far. I wish even more people could be as privileged as I have been throughout all these years, so that we could build together a better country for everyone.

Fortunately, the list of friends and colleagues that I would like to thank is quite large. The path taken between the first time I considered doing this doctorate until its completion was very long, which made me come across many people who, in different ways, contributed to this achievement. Due to space constraints and to avoid forgetting to thank someone, I would just like to extend my most sincere gratitude for the technical, scientific, and, above all, nontechnical contributions made by everyone. Friends and colleagues from POLI-UPE, CIn-UFPE and ERCIS-WWU, as well as the friends I keep outside the academic world, without your support, the accomplishment of this mission would have been much more difficult.

To Prof. Teresa Ludermir and Prof. Herbert Kuchen, who gave me the opportunity to absorb their knowledge and learn from their vast experience during the last 4 years, thank you very much for trusting my work and my potential. I feel privileged to have had the chance to be supervised by world-class advisors like you both. To my academic father Prof. Fernando Buarque, who led me and shaped me academically since my scientific initiation in 2009, from whom I have enormous affection and admiration, thank you very much for everything. I will always carry your academic genes with me, which I hope to honor during my life as a researcher. I hope I can return to society the privilege of having been educated by you.

I would like to thank my uncles, aunts, and cousins for all the support given during the most difficult moments of this journey. I would like to thank my grandfather Moacir and my grandmother Dalila for having given all the care and support during this phase. However, I would like to especially thank my grandmother Aliete, who unfortunately could not witness the end of this long journey. Her joy with each achievement of mine, from the beginning of my life until our last moments together, will be forever in my mind and in my heart.

I would like to give special thanks to my brother Bruno, my sisters-in-law Rhanna and Camylle, and my mother-in-law Socorro for all the moments together. Such moments really helped me to take part of the worries out of my mind in the most difficult moments. You were essential to the success of this doctorate. However, I would like to give special thanks to my parents Jorge and Josélia, who taught me from the first moments of my life the importance of knowledge, honesty, and, above all, correctness in my decisions. You both are the best references of good human beings that I could have. Thank you very much for having shown me since I was a child the wonders of discovering something new, of exploring the unknown. Without the countless hours that you both dedicated to my education, and without the good emotional structure kept at home during my entire life, I would certainly be in a completely different situation now. If I am very happy today personally and professionally, since I do what I love, that is because of you, and I will owe you forever. Thank you very much.

Last, but definitely not least, I would like to thank my dear wife, Caroline, for all these years by my side, giving all the necessary support so that I could achieve this goal. Thank you very much for holding my hand during all the moments of abdication, for always being there for me, for making me believe that everything would work out in the end. Thank you very much for your patience at the moments when I needed to isolate myself from everything and everyone to focus on this research. Thank you very much for the words of encouragement, for all the care and love dedicated during all these years, doing the possible and impossible so that I could be truly happy and successful in my profession. Thank you very much for always supporting me in the most critical decisions, for always being with me, no matter what, and for keeping everything in order at home in the most critical moments of this journey, often abdicating your personal and professional life. Be sure that we have built this together. Without your love, your support and your understanding, this achievement would not be possible. Finally, thank you very much for making the wise decision to bring our "four-legged child" into our lives in the last year of this journey, which brought much more joy to my days and was essential during the last year, which was unique and very difficult for everyone.

I hope I have honored the support given by all of you during this long journey. To all of you, the deepest and most sincere gratitude.

ABSTRACT

Despite the success of evolutionary and swarm-based algorithms in many different application areas, such algorithms are very sensitive to the values of their parameters. According to the No Free Lunch Theorem, there is no parameter setting for a given algorithm that works best for every possible problem. Thus, finding a quasi-optimal parameter setting that maximizes the performance of a given metaheuristic in a specific problem is necessary. As manual parameter adjustment for evolutionary and swarm-based algorithms can be very hard and time demanding, automating this task has been one of the greatest and most important challenges in the field. Out-of-the-box parameter control methods are techniques that dynamically adjust the parameters of a metaheuristics during its execution and can be applied to any parameter, metaheuristic and optimization problem. Very few studies about out-of-the-box parameter control methods can be found in the literature, and most of them apply reinforcement learning algorithms to train effective parameter control policies. Even though these studies have presented very interesting and promising results, the problem of parameter control for metaheuristics is far from being solved. A few important gaps were identified in the literature of this field, namely: (1) training parameter control policies with reinforcement learning can be very computational-demanding; (2) reinforcement learning algorithms usually require the adjustment of many hyperparameters, what makes difficult its successful use. Moreover, the search for an optimal policy can be very unstable; (3) and, very limited benchmarks have been used to assess the generality of the out-of-the-box methods proposed so far in the literature. To address such gaps, the primary objective of this work is to propose an out-of-the-box policy training method for parameter control of mono-objective evolutionary and swarm-based algorithms with distributed reinforcement learning.The proposed method had its generality tested on a comprehensive experimental benchmark with 133 scenarios with 5 different metaheuristics, solving several numerical (continuous), binary, and combinatorial optimization problems. The scalability of the proposed architecture was also dully assessed. Moreover, extensive analyses of the hyperparameters of the proposed method were performed. The experimental results showed that the three aforementioned gaps were successfully addressed by the proposed method, besides a few other secondary advancements in the field, all commented in this thesis.

Keywords: swarm intelligence; evolutionary computation; reinforcement learning; parameter control.

RESUMO

Apesar do sucesso de algoritmos evolutivos e baseados em enxames em diferentes áreas de aplicação, estes algoritmos são muito sensíveis aos seus parâmetros. De acordo com o teorema "não existe almoço grátis", não existe configuração para um determinado algoritmo que funcione melhor para todos os problemas possíveis. Assim, faz-se necessário encontrar uma configuração de parâmetro que maximize o desempenho de uma dada metaheurística em um problema específico. No entanto, o ajuste manual de parâmetros para algoritmos evolutivos e baseados em enxames pode ser muito difícil e exigir muito tempo. Portanto, automatizar essa tarefa tem sido um dos maiores e mais importantes desafios da área. Métodos out-of-the-box de controle de parâmetros são técnicas que ajustam dinamicamente os parâmetros de uma metaheurística durante sua execução e podem ser aplicados a qualquer parâmetro, metaheurística e problema de otimização. Poucos estudos sobre métodos de controle de parâmetros out-of-the-box podem ser encontrados na literatura, e a maioria deles aplica algoritmos de aprendizagem por reforço para treinar políticas de controle de parâmetros eficazes. Embora esses estudos tenham apresentado resultados muito interessantes e promissores, o problema do controle de parâmetros para metaheurísticas está longe de ser resolvido. Algumas lacunas importantes foram identificadas na literatura da área, a saber: (1) Métodos de treinamento de políticas de controle de parâmetros baseados em aprendizagem por reforço podem demandar muito esforço computacional e tempo de execução. (2) Algoritmos de aprendizagem por reforço geralmente requerem o ajuste de vários hiperparâmetros, o que dificulta seu uso com sucesso. Além disso, a busca por uma política ótima pode ser muito instável. (3) Benchmark experimentais muito limitados foram usados para avaliar a generalidade dos métodos out-of-the-box, o que limita a avaliação da generalidade dos métodos propostos. A fim de preencher tais lacunas, o objetivo principal deste trabalho é propor um método de treinamento de política out-of-the-box para controle de parâmetros de algoritmos evolucionários e baseados em enxames mono-objetivos utilizando aprendizagem por reforço distribuída. A fim de avaliar sua generalidade, o método proposto foi testado em um benchmark experimental abrangente com 133 cenários com 5 metaheurísticas diferentes, resolvendo vários problemas de otimização contínua, binários e de otimização combinatória. A escalabilidade da arquitetura proposta também foi avaliada. Além disso, foi realizada uma análise dos hiperparâmetros do método proposto. Os resultados experimentais mostraram que as três lacunas acima mencionadas foram satisfatoriamente preenchidas pelo método proposto, além de alguns outros avanços secundários na área.

Palavras-chave: inteligência de enxames, computação evolucionária, aprendizagem por reforço, controle de parâmetros.

LIST OF FIGURES

LIST OF TABLES

CONTENTS

1 INTRODUCTION

"Imagination is the Discovering Faculty, pre-eminently. It is that which penetrates into the unseen worlds around us, the worlds of Science."

Ada Lovelace

1.1 OPTIMIZATION AND METAHEURISTICS

Optimization is the process of finding a set of arguments that maximizes or minimizes a mathematical function. Mathematically, optimization is the minimization or maximization of an objective function f subject to constraints c_i on its vector of variables $\mathbf{x}$, as shown in Equation 1.1 (NOCEDAL; WRIGHT, 2006; ENGELBRECHT, 2007), where E and I are the sets of equality and inequalities constraints.

$$\min_{\mathbf{x}\in\mathbb{R}^n} f(\mathbf{x}), \tag{1.1}$$

subject to

$$c_i(\mathbf{x}) = 0, i \in E, \tag{1.2}$$

and

$$c_i(\mathbf{x}) \geq 0, i \in I. \tag{1.3}$$

An optimization problem can be seen as a search problem, since they can be solved by searching for the appropriate solution among a set of possible ones, known as the search space (NOCEDAL; WRIGHT, 2006). Due to the size and dimensionality of the search space or to the characteristics of its surface (*e.g.* whether the objective function is convex or not), many optimization problems are in NP, but not in P, or even in NP-hard. It means that even though a solution for such problems can be verified in polynomial time, the most efficient algorithm known so far solves them at least in exponential time (FORTNOW, 2009). It is important to mention that if $P = NP$, the NP problems outside NP-hard can be solved in polynomial time. However, by the time this thesis was written, P has not been proven to be equals NP. For these problems, finding an exact solution to large instances might take an unacceptable amount of time. Thus, approximate methods should be considered (EIBEN; SMITH, 2015; TALBI, 2009).

Heuristics are approximate search methods that, following a set of previous assumptions (*i.e.* previous human knowledge) find good solutions without trying every possible solution in the search space. The drawback of these techniques is that they do not guarantee the best solution possible. However, they are able to find good enough solutions in polynomial time (FOULDS, 1983; SILVER, 2004).

Heuristics can be divided into problem-specific heuristics and metaheuristics. Surely, problem-specific heuristics are approximate search methods that use rules designed to solve specific problems. On the other hand, metaheuristics are designed to be applied to many different search problems (TALBI, 2009).

Metaheuristics can be divided into single-solution-based and population-based algorithms (PBA). Single-solution-based methods use a single worker that iteratively tests sequential solutions in a trial-and-error mechanism. Population-based algorithms use a population of workers that test multiple solutions in parallel. These workers must communicate with each other to improve their search capabilities (TALBI, 2009).

Evolutionary algorithms (EA) and swarm-based algorithms (SI) are two distinct families of PBAs. EAs are inspired by biological genetics and natural selection (EBERHART, 2007), while swarm-based algorithms are inspired by the collective behavior of animals in nature. Due to the interaction between the workers that represent the population of solutions, swarm-based algorithms exhibit a property called Swarm Intelligence. Swarm Intelligence is the property of any system composed by numerous simple parts that interact with each other, emerging complex patterns from such interactions (BONABEAU; DORIGO; THERAULAZ, 1999; PANIGRAHI; SHI; LIM, 2011).

1.2 PARAMETER ADJUSTMENT FOR EA AND SWARM-BASED ALGORITHMS

Despite the success of EA and swarm-based algorithms in many different application areas (ALETI; MOSER, 2016), such algorithms are very sensitive to the values of their parameters (EIBEN; SMIT, 2011). Two kinds of parameters of metaheuristics can be distinguished:

1. Numerical parameters: Parameters that assume only integer or real values.
2. Categorical parameters: Parameters that assume categorical values, *i.e.* unordered values taken from a finite set of discrete values. Usually these parameters determine the logic that will be used in the operators of the metaheuristic.

According to the No Free Lunch Theorem (NFL), there is no parameter setting for a given algorithm that works best for every possible problem (WOLPERT; MACREADY, 1997). Thus, finding an optimal parameter setting that maximizes the performance of a given metaheuristic in a specific problem is necessary. However, manual parameter adjustment for EA and SI can be very hard and time demanding. Therefore, automating this task has been one of the greatest and most important challenges in the field (EIBEN; HINTERDING; MICHALEWICZ, 1999).

Autonomous adjustment approaches can be divided into two groups (KARAFOTIAS; HOOGENDOORN; EIBEN, 2015b):

- Tuning algorithms: The parameters are adjusted before the execution of the algorithm, usually via multiple previous runs. These techniques identify the best set of values that maximizes the performance of a given algorithm for the problem at hand. The chosen values are kept constant during the entire execution of the optimization process.

- Control algorithms: The parameters are adjusted on-the-fly, *i.e.* during the execution of the algorithm. Control methods choose a set of values for the optimization algorithm, so that it runs for a given amount of time and, then, returns a performance measure. The controller is therefore able to know how good that choice was. These steps are repeated over and over, always trying to maximize the performance of the optimization algorithm by making the best choice for each moment of the optimization process.

The techniques of both groups aim at reducing the need for manual adjustment or, in a few cases, completely remove such a need. Such a reduction is achieved by substituting the original set of parameters of the optimization algorithm by a smaller set added by the controller, or by an equally large but less sensitive set of parameters. The complete removal of manual parameter adjustment is achieved by very few techniques, which is usually done by freezing "parameters" to values that present usually good results (PARPINELLI; PLICHOSKI; SILVA, 2019). However, it is well-known that the optimal value of a given parameter changes along the optimization process, and only control methods are able to perform such a dynamic adjustment (EIBEN; HINTERDING; MICHALEWICZ, 1999; KARAFOTIAS; HOOGENDOORN; EIBEN, 2015b).

From the perspective of machine learning, parameter control can be understood as the learning problem of finding a good policy to dynamically adjust the parameters of an algorithm for a set of unseen target problems. This learning process is performed over a set of training instances. Even though such a perspective was initially defined to be applied to parameter tuning and has been used since then (LóPEZ-IBáñEZ et al., 2011; BIRATTARI et al., 2010), it can be easily transferred to parameter control (BIRATTARI, 2009).

From now on, in this work, whenever the word "parameter" is used, the class of numerical parameters is referred. If categorical parameters had also been considered, operator selection methods (*i.e.* methods that dynamically change the operators of a running metaheuristic) would have to be considered as parameter control techniques, which is not considered in this thesis. The adjustment of categorical parameters is often called *Adaptive Operator Selection*, and the algorithms that present such a feature are usually called *hyperheuristics* (CONSOLI et al., 2016; DACOSTA et al., 2008; CONSOLI; MINKU; YAO, 2014; DRAKE et al., 2020). Therefore, there is a well-established field already existent that is out of the scope of this study.

Among the control methods, two subgroups can be identified (KARAFOTIAS; HOOGENDOORN; EIBEN, 2015b):

- Control methods tailored to a specific application: In this group, the control methods are conceived to work with specific algorithms, controlling specific parameters and solving specific problems.
- Out-of-the-box control methods: These control methods can be applied to any optimization algorithm, parameter and/or optimization problem.

The vast majority of the studies about parameter control methods for EA and SI that have been published so far focus on the methods tailored to specific applications. A few publications have been made on the development of out-of-the-box methods. Surveys on the topic of parameter adjustment have been published by Eiben *et al.* in 1999 (EIBEN; HINTERDING; MICHALEWICZ, 1999), Zhang *et al.* in 2012 (ZHANG et al., 2012), Karafotias *et al.* in 2015 (KARAFOTIAS; HOOGENDOORN; EIBEN, 2015b), Aleti *et al.* in 2016 (ALETI; MOSER, 2016), Guan *et al.* in 2017 (GUAN; YANG; SHENG, 2017) and Parpinelli *et al.* in 2019 (PARPINELLI; PLICHOSKI; SILVA, 2019).

In 2021, a systematic literature review about out-of-the-box parameter control methods for EA and swarm-based algorithms was produced (LACERDA et al., 2021). In this study, it was revealed that most of the papers about out-of-the-box parameter control for metaheuristics have applied Reinforcement Learning (RL) (EIBEN et al., 2007; KARAFOTIAS; SMIT; EIBEN, 2012) (KARAFOTIAS; EIBEN; HOOGENDOORN, 2014; KARAFOTIAS; HOOGENDOORN; WEEL, 2014; KARAFOTIAS; HOOGENDOORN; EIBEN, 2015a; ROST; PETROVA; BUZDALOVA, 2016). Other papers have applied different techniques to create controllers, such as regression methods in order to predict the performance of the metaheuristic given different configurations and scenarios, among others (ALETI et al., 2014; ALETI; MOSER, 2013; BIELZA; POZO; LARRAñAGA, 2013; ALETI; MOSER; MOSTAGHIM, 2012; LEUNG; YUEN; CHOW, 2012) (CHATZINIKOLAOU, 2011; ALETI; MOSER, 2011; MATURANA; SAUBION, 2008) (AINE; KUMAR; CHAKRABARTI, 2006). This study will be detailed later in this work.

RL algorithms are machine learning techniques where an interacts with its surrounding environment by taking actions and receiving rewards, improving its performance by trial and error (SUTTON; BARTO, 2018). They are especially useful when decisions must be taken over time and the training data is not independent and identically distributed (i.i.d.), which is the case of parameter control, since a scenario at a given time t can be highly correlated with a scenario at $t+1$, for instance. However, even though the RL-based out-of-the-box methods have presented the most promising results so far, the solutions for the problem of parameter control for metaheuristics still has a lot to be improved. In our literature review, we have identified a few issues that are still open. Three of them, dealt by this thesis, are listed below:

1. Training EA and SI parameter control policies with RL can be very computational-demanding. And to date no study has ever proposed a scalable approach that could benefit from parallel and distributed computing platforms.

2. RL algorithms usually require the adjustment of many hyperparameters, what makes difficult its general use. Also, the search for an optimal policy can be very unstable, since RL algorithms usually suffer from bias overestimation caused by the "dog chasing its tail" effect in the Bellman Equations. It means that, for many algorithms, the policy search is likely to exploit wrong decisions and prone to get stuck in local minima (FUJIMOTO; HOOF; MEGER, 2018).

3. Even though the authors of the reviewed studies argue that their proposed methods are out-of-the-box, very limited benchmarks have been used to assess such a generality, what reduces the scope and generality of these methods.

1.3 OBJECTIVES

1.3.1 General Objective

The primary objective of this work is to propose an out-of-the-box policy training method for parameter control of mono-objective EA and swarm-based algorithms with distributed Reinforcement Learning, addressing the three aforementioned issues identified in the literature.

1.3.2 Specific Objectives

Each addressed issue is related to a specific objective in this work, which are listed below:

1. Propose a scalable parameter control policy training method that produces a policy that can be succesfully used in an unseen problem.

2. Propose a mechanism that diminishes the difficulties on hyperparameter adjustment and instability of the learning process.

3. Assess the generality of the proposed method in an experimental benchmark with metaheuristics and optimization problems with different characteristics, including EA and swarm-based algorithms solving numerical and categorical optimization problems.

1.4 THESIS STRUCTURE

The remaining of this work is organized as follows: Chapter 2 presents the Background needed to follow the proposed core ideas of this thesis; Chapter 3 presents the systematic literature review; Chapter 4 describes the proposed method; Chapter 5 details the experimental methodology and discusses the results achieved in the experiments carried out in this study; and Chapter 6 presents the conclusions drawn from this study. As this thesis rely heavily in experimental work, the figures and tables resulting of the extensive simulations are presented in Appendices A1-A7, just after the reference section. Appendix A8 contains title and abstracts of all the articles written by the author during the research that are related to the study carried out.

2 BACKGROUND

"(...) in my opinion, all things in nature occur mathematically."

Rene Descartes

In this chapter, the theoretical background that is necessary to understand this thesis is presented.

2.1 BASIC CONCEPTS OF METAHEURISTICS

2.1.1 Classical Optimization Models and Exact Optimization Methods

In search problems with feasible solutions, there is a solution in an unknown location in a search space that must be found. Optimization and modelling are two families of search problems. Modelling problems involve finding a representation of an observed phenomenon that produces the correct outputs for the observed inputs (EIBEN; SMITH, 2015). As previously defined, optimization is the process of finding a set of arguments that maximizes or minimizes an objective function. In a more practical point of view, optimization is the task of taking decisions that maximize the performance of the decision maker in a given task.

The process to solve an optimization problem can be divided into 4 steps: problem formulation, problem modelling, problem optimization (solving), and solution implementation. The problem modelling phase is essential since it determines realistic is your model, and therefore how useful your solution is in the real world. Also, it determines the optimization method to solve the problem (TALBI, 2009).

An optimization problem can be defined as a tuple (S, f), where S represents the set of all feasible solutions, and f is the objective function $f : S \longrightarrow \mathbb{R}$ that determines the quality of each feasible solution. A global optimum is a solution $\mathbf{s}^* \in S$, $\forall \mathbf{s} \in S, f(\mathbf{s}^*) \leq f(\mathbf{s})$ if f must be minimzed, and $\mathbf{s}^* \in S$, $\forall \mathbf{s} \in S, f(\mathbf{s}^*) \geq f(\mathbf{s})$ otherwise. In other words, a global optimum is a feasible solution that is no worse than any other. It means that there might be many global optima in a single function. The best solution in a limited area of the search space that is worse than the global optima is called local optimum. The objective of any optimization method is to find the global optima of a given optimization problem (TALBI, 2009).

There are several families of optimization models used to formulate optimization problems. The two most common approaches to build and optimization model is mathematical programming and constraint programming (TALBI, 2009). In constraint programming, the solution is defined purely by constraint functions. It means that the solution of the problem is found by satisfying all constraints. In mathematical programming, which is the most common approach for building optimization models, the search process is guided by an objective function (EIBEN; SMITH, 2015). There are also cases where constraints are used to limit the search space represented by an objective function. This work focuses on optimization methods for mathematical programming models. Thus, constraint programming-based approaches will not be further detailed. Also, in the real world, most of the optimization problems present constraints. Thus, constraint functions will always be considered in the definitions presented in this background section. In these cases, the reader might consider the existence of an empty set of constraints.

A widely used mathematical programming model is linear programming (LP). In LP, both the objective function and constraints are linear continuous functions. In such problems, the region of feasible solutions is a convex set and the objective function is convex. Thus, exact approaches such as the Dantzig's Simplex algorithm (DANTZIG, 1961) solve them efficiently (*i.e.* in polynomial time) (TALBI, 2009).

Another mathematical programming model used for continuous optimization is nonlinear programming (NLP). In such models, the objective function and/or the constraints are nonlinear continuous functions (BERTSEKAS, 1999). As opposed to linear programming models, NLPs are difficult to solve. Linearizing the model is one alternative to make it solvable by an exact algorithm for LP models. However, the linearization of nonlinear models always shows approximation errors. For quadratic or other convex functions, exact algorithms can be used within an acceptable execution time in moderate problems (NOCEDAL; WRIGHT, 2006). However, for problems with high dimensionality, non-separable dimensions, multimodality and nondifferentiability, approximate methods are mandatory (TALBI, 2009).

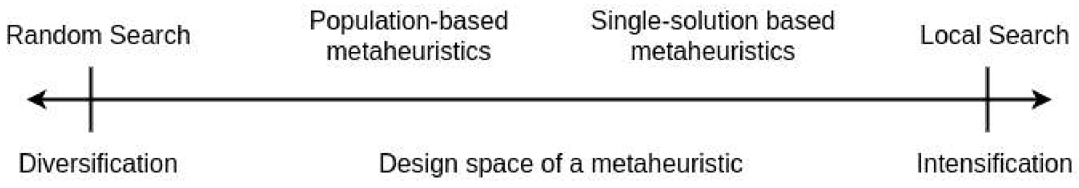
For numerical problems with discrete variables, integer programming (IP) can be used for modelling. When the problem presents both discrete and continuous variables, mixed integer programming models (MIP) can be applied. A more general type of IP problems is the family of combinatorial optimization problems, of which variables have a finite set of discrete values. Usually, there is no order among such values. The majority of the real world combinatorial problems are not solvable by an exact algorithm in polynomial time (at least so far) and many of them are NP-Hard (TOSCANI; VELOSO, 2012). Therefore, approximate methods can be used to give satisfactorily good solutions (*i.e.* optimal solutions are not guaranteed).

2.1.2 Metaheuristics

For NP-Hard problems, heuristics are good options as approximate search methods. However, heuristics that are designed for these problems are usually ineffective in instances with higher dimensionality. Moreover, heuristics are approximate solutions conceived for specific problems, which means that a heuristic designed for a given problem A will not work in a different problem B, unless B is polynomially reduceable to A (TALBI, 2009).

Metaheuristics are heuristics designed to solve multiple problems. They have received an increasing attention in the last 30 years due to its ability in approximate the solution of large-sized problems within a reasonable time. There is a classic trade-off that must be considered when a new metaheuristic is conceived or an already existing one is used to solve a problem: exploration versus exploitation. Exploration is the ability of the metaheuristic to diversify its search process, searching for solutions in a wide area in the search space. On the other hand, exploitation is the capacity of intensification of the search process in a promising region. In Figure 1, the design space of a metaheuristic is presented ranging from random search, that is pure diversification (*i.e.* exploration), to local search, which is total intensification (*i.e.* exploitation). Random search consists in randomly generating a new solution at each iteration (*i.e.* no memory is used), while local search generates a new solution by selecting the best neighbor to the current solution (TALBI, 2009).

Figure 1 – Conflicting criteria in designing a metaheuristic: exploration versus exploitation.


Random Search
Population-based metaheuristics
Single-solution based metaheuristics
Local Search
Diversification
Design space of a metaheuristic
Intensification

Source: Produced by the author based on (TALBI, 2009).

Between random search and local search, the metaheuristics can be divided into population-based (*e.g.* Ant Colony Optimization (ACO) (DORIGO, 1992), Artificial Bee Colony (ABC) (KARABOGA; BASTURK, 2008), Crow Search Algorithm (CSA) (ASKARZADEH, 2016), Differential Evolution (DE) (STORN; PRICE, 1997), Elephant Herding Optimization (EHO) (WANG; DEB; COELHO, 2015), Fish School Search (FSS) (FILHO et al., 2009a), Genetic Algorithms (HOLLAND, 1975), Particle Swarm Optimization (PSO) (KENNEDY; EBERHART, 1995)), and single-solution based metaheuristics (*e.g.* Guided Local Search (GLS) (VOUDOURIS, 1998), Greedy Adaptive Search Procedure (GRASP) (FEO; RESENDE, 1989), Iterated Local Search (ILS) (MARTIN; OTTO; FELTEN, 1991), Simulated Annealing (SA) (CERNY, 1985), Tabu Search (TS) (GLOVER, 1986). As explained in the previous chapter, single-solution-based metaheuristics are based on the idea of having a single solution evolving throughout the iterations of the search process, while PBAs have multiple solutions evolving in parallel. As expected, multiple solutions bring more diversification to the search process than a single evolving solution.

According to Talbi (TALBI, 2009), metaheuristics can also be classified according to the following aspects:

- Nature inspired versus non-nature inspired metaheuristics: Metaheuristics can be inspired by mechanisms in nature, like the collective behavior of animals or the natural evolution of species.
- Memory-based versus memoryless methods: As previously mentioned, when local search was compared against random search, metaheuristics can make use of memory (*i.e.* use the search history to make better decisions in the future) or be completely memoryless.
- Deterministic versus stochastic metaheuristics: Deterministic metaheuristics produce the same final solution for the same initial solution, while stochastic metaheuristics present some degree of randomness, which causes the algorithm to return different solutions even though the same initial solutions are given.
- Iterative versus greedy methods: Iterative metaheuristics start with single or multiple solutions that are transformed throughout the iterations by search operators, while greedy algorithms assign values to each of the decision variables one by one, until a complete solution is obtained. It is important to highlight that the vast majority of metaheuristics are iterative

2.2 EVOLUTIONARY AND SWARM-BASED ALGORITHMS

Evolutionary and Swarm-based algorithms are stochastic, iterative nature inspired metaheuristics. They mostly use memory to guide their search process. This section presents the most important aspects of both families of metaheuristics and describes a few relevant algorithms, which were used in the experiments presented later in this work.

2.2.1 Evolutionary Algorithms

Evolutionary Algorithms are metaheuristics inspired by the process of natural evolution. Such mechanisms are adapted and simplified to fit to the trial-and-error dynamics that are present in every metaheuristic. The first ideas of applying Darwinian principles to automated problem solving date back to the 1940s, when Alan Turing proposed "genetical or evolutionary search" (EIBEN; SMITH, 2015). However, the first implementation of an optimization algorithm based on evolution and recombination mechanisms was first executed in a computer in 1962 (EIBEN; SMITH, 2015). During the 1960s and 1970s, three different implementations of the basic ideas of evolutionary search started to be developed independently: Evolutionary Programming (EP) (FOGEL; OWENS; WALSH, 1966), Genetic Algorithms (GA) (JONG, 1975; HOLLAND, 1973), and Evolution Strategies (ES) (RECHENBERG, 1973). In the early 1990s, these three approaches became part of a new field called Evolutionary Computing (EC), of which algorithms were termed Evolutionary Algorithms (EA) (EIBEN; SMITH, 2015). Then, these algorithms were divided into the three already mentioned groups EP, GA and ES, and a fourth one called Genetic Programming (GP), that was first proposed in the 1990s (KOZA, 1990).

Every EA is based on the following idea: given a population of candidate solutions living in an environment with limited resources, they must compete for those resources to survive. Such a competition causes the so-called natural selection. Thus, the fittest ones survive. The general scheme of EAs is presented in the Algorithm 1.

Algorithm 1: General scheme of an Evolutionary Algorithm.

```
  Input: Randomly initialized population of candidate solutions
  Output: Best solution ever found
1 EVALUATE each candidate
2 while Stopping condition is not met do
3     SELECT parents
4     RECOMBINE pairs of parents
5     MUTATE the resulting offspring
6     EVALUATE new candidates
7     SELECT candidate solutions for the next generation
8 return Best solution ever found
```

In Algorithm 1, it can be noticed that the basic scheme of an EA is composed by four operators: recombination, mutation, evaluation and selection. These operators are executed for each iteration (*i.e.* generation). It starts with the selection of parents for the recombination step. Then, the offspring generated during the recombination procedure is mutated. After that, the new candidates are evaluated and, finally, the surviving candidate solutions are selected to pass to the next generation. The mechanisms implemented in each operator vary according to the algorithm. Thus, before using or creating an EA, these operators must be defined. Besides, the structure of the candidate solutions that form the population of solutions must be decided beforehand.

As previously mentioned, each candidate solution represents a candidate solution to the optimization problem at hand. Therefore, its structure depends on the problem, which is modeled through an objective function, and a set of constraint functions. For example, a binary vector represents an candidate solution (*i.e.* solution) in a binary optimization problem, while a real-coded vector represents a solution to a numerical optimization problem modeled as a mathematical function with continuous variables. Thus, after modeling the problem using one of the optimization models presented earlier in this work, the solution representation (*i.e.* structure of candidate solutions) must be defined.

Defining the representation of solutions, the aforementioned four operators can be defined. The selection mechanism involves the procedure used to select the surviving candidate solutions from a population to pass to a new generation (*i.e.* iteration). This step can be called *replacement*. Also, this mechanism can be used to select candidate solutions to recombine or mutate, *i.e.* generate new candidate solutions. It is important to mention that the best solutions are more likely to be chosen in the selection process, and the higher such a probability, the stronger the evolutionary pressure in the population over the generations.

The recombination mechanism, also known as crossover, involves the exchange of information (*i.e.* values from the solution vectors) between selected candidate solutions, while the mutation procedure generates a random perturbation in the solution vectors. The first mechanism causes the algorithm to exploit promising regions, while the second one generates diversity in the search process.

Finally, the evaluation step is the part of the algorithm where the quality of each surviving solution is measured. This mechanism uses the objective function to evaluate each candidate solution in the population, assigning a numerical value to each one of them. Such a value is called *fitness* and it is used to identify the fittest candidate solutions. The fittest candidate solutions are the ones with the highest probability of surviving over the iterations and passing their genes through generations via successive recombinations (EIBEN; SMITH, 2015).

There are several possibilities for the stopping condition. Its choice depends on the necessities of the user or designer of the algorithm. For example, the iterative algorithm might stop when it reaches a given number of iterations, or when its best solution does not improve anymore for a given number of iterations.

In the following sections, two EAs will be detailed: Genetic Algorithms and Differential Evolution (DE). These methods have been chosen to be presented in this section because they have been used in the experiments of this study.

2.2.1.1 Genetic Algorithms

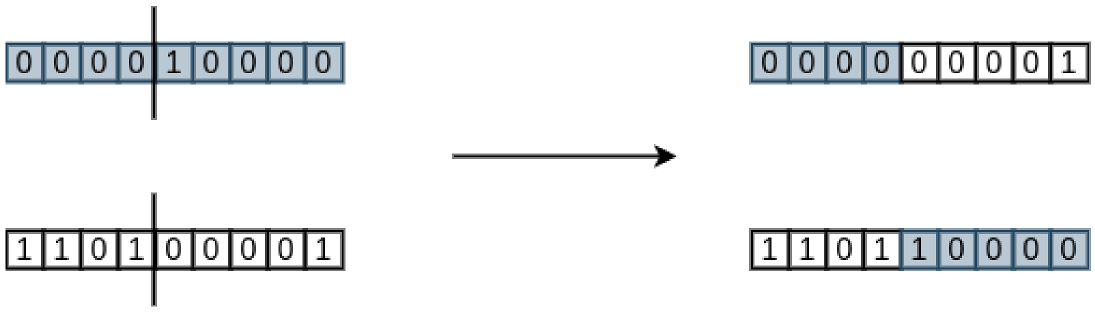
The basis for the Genetic Algorithms was initially established by Holland as a means of studying the adaptive behavior in nature (HOLLAND, 1973). However, the studies of De Jong (JONG, 1975) and Goldberg (GOLDBERG, 1989) greatly contributed to define what came to be the simple (or canonical) version of the GA as an optimization algorithm. Its operators and solution representation are described below.

In the simple GA, the candidate solution $\mathbf{x}$ is a binary-coded solution vector, *i.e.* $\mathbf{x} \in \{0, 1\}^n$, where n is the number of dimensions of the binary decision space. It means that such an algorithm is more suitable to solve binary optimization problems, which can be modeled as an Integer Programming model where the variables range between 0 and 1 in the feasible space. The evaluation of a solution in the simple GA (considering an IP model for the problem at hand) is straightforward, since each bit in the bit string (*i.e.* each gene in the chromosome) represents one variable in the decision space.

There are three standard methods for recombination of binary-encoded solution vectors: one-point crossover, n-point crossover, and uniform crossover. For all methods, two parents are recombined to create two children. In the one-point crossover, two parents exchange their genes by randomly selecting a random number within the range $[1, n-1]$, where n is the number of bits in the parent vectors. Then, both parents are splitted at this point and two children are created by exchanging their tails.

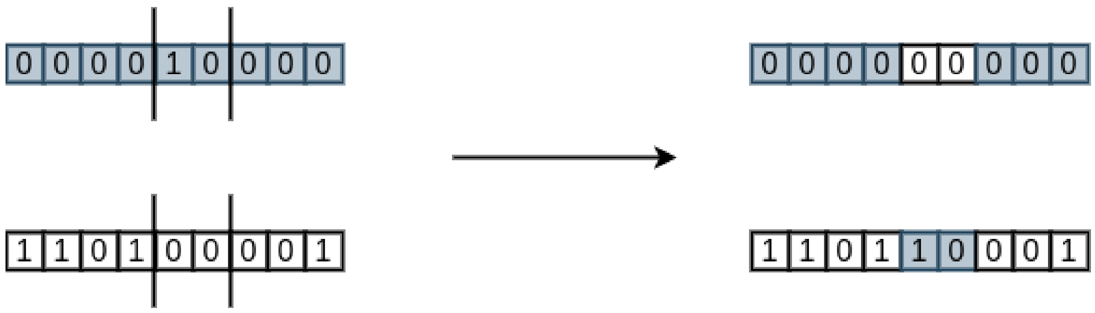
The n-point crossover is a generalization of the one-point crossover, where n points are randomly selected instead of one, and the parts between the splitting points are exchanged between the parents. Finally, in the uniform crossover, n random numbers between 0 and 1 are generated, one for each variable in the solution vector. For the variables of which the corresponding random values lie below 0.5, the value of the first parent will be copied into the first child, and the value of the second parent will be copied into the second child. Otherwise, the bit from the first parent is copied into the second child, and the bit from the second parent is copied into the first child (EIBEN; SMITH, 2015). The one-point, n-point and uniform crossover methods are illustrated in Figures 2, 4 and 4, respectively. The crossover operator is executed with a given probability p_c.

Figure 2 – Example of the one-point crossover method.



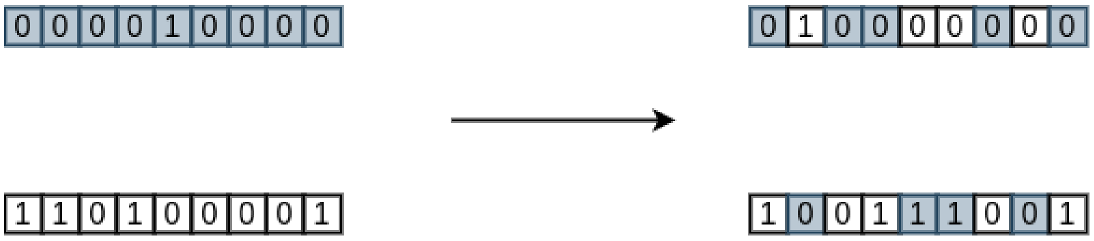
Source: Produced by the author based on (EIBEN; SMITH, 2015).

Figure 3 – Example of the n-point crossover method, where $n = 2$.



Source: Produced by the author based on (EIBEN; SMITH, 2015).

Figure 4 – Example of the uniform-point crossover method. The array [0.3, 0.6, 0.1, 0.4, 0.8, 0.7, 0.3, 0.5, 0.3] of random numbers were used to decide the inheritance.



Source: Produced by the author based on (EIBEN; SMITH, 2015).

The standard mutation process for binary strings is quite simple. Each bit of each candidate solution flips with some probability. When a bit is flipped, it turns 1 if its previous value is 0, and vice-versa. Surely, the mutation probability must be low in order to avoid too much randomness in the search process (EIBEN; SMITH, 2015).

Binary-encoded solutions can also be used to solve nonbinary optimization problems. For example, let $\mathbf{x}$ be a binary-encoded solution vector with 80 bits, where each subset of contiguous 8 bits represents an integer variable in the decision space of an optimization problem. It means that the given problem has 10 decision variables and each variable is able to represent 256 possible values. In some cases, the binary strings are also used to represent a real-valued vector. Considering the 80 bits-long string and a floating-point representation of real values with 16 bits, the optimization problem has a decision space with 5 real-valued decision variables. However, a few problems arise when such an approach is used. One of them is the difference of significance between each bit, what makes the probability of changing the value of a variable from 2 to 3 the same as changing it from 0 to 128. Another issue is that the Hamming distance between two consecutive integers is not always 1 (*e.g.* the Hamming distance between 2 and 3 is indeed 1, but the distance between 127 and 128 is 8). Thus, due to the mentioned issues, it is recommended to use a proper integer or real-valued representation for the solutions vectors in such optimization problems. Surely, the operators must be defined accordingly (EIBEN; SMITH, 2015).

The selection operator is responsible for the evolutionary pressure on the population (*i.e.* exploitation). It keeps the fittest candidate solutions alive as long as possible and assures that their genes are likely to be passed over the generations. Selection procedures can be used both for parent selection and survival selection. It is important to highlight that some of the selection mechanisms available in the literature are common to many EAs. In order to illustrate how selection methods work, a few selection approaches are detailed below.

As already mentioned, in parent selection operators, a set of λ candidate solutions are selected and copied into a mating pool. It is possible to enable the replacement of the selected candidate solutions, which means that there may be multiple copies of the best candidate solutions in the pool, what increases the evolutionary pressure. Then, pairs of candidate solutions are randomly selected to recombine their genes. Three approaches are widely used in the literature: fitness proportional selection (FPS), ranking selection (RS), and tournament selection (TS).

For a given candidate solution i, the fitness proportional selection assigns a selection probability proportional to its fitness, as described in the Equation 2.1, where f_i is the fitness of candidate solution i and μ is the population size. This method was introduced in (HOLLAND, 1992) and, since then, it has been widely studied, especially due to the fact that it is easier to perform theoretical analysis on it. However, it is known that its canonical form presents a few critical drawbacks, such as its strong tendency to premature convergence, especially when there is a small group of outstanding solutions in comparison to the others, and its low evolutionary pressure when the quality of the solutions are too close to each other (EIBEN; SMITH, 2015).

$$P_{FPS}(i) = \frac{f_i}{\sum_{j=1}^{\mu} f_j}. \tag{2.1}$$

The ranking selection method was conceived in an attempt to surpass the issues of premature convergence and low evolutionary pressure of the FPS method. Proposed in (BAKER, 1987), it keeps the evolutionary pressure constant during the search process by defining the selection probability inversely proportional to its rank in a population sorted by the fitnesses of the candidate solutions in descending order. Such a principle can be implemented in many ways. For example, the most common and simple inversely proportional function is linear, as described in Equation 2.2, where s $(1 < s \leq 2)$ is a parameter of the operator, i is the ranking of the candidate solution. In such an implementation, the best candidate solution has rank $\mu - 1$ and the worst has rank 0. It is also possible to put more evolutionary pressure in the search process by using an exponential function instead of the linear one, where the difference between the selection probabilities of the top-ranked candidate solutions and the remaining ones is greater than in the linear approach. Such an approach is defined in the Equation 2.3, where c must be defined so that the sum of the probabilities is equal to 1 (EIBEN; SMITH, 2015).

$$P_{lin-rank}(i) = \frac{2-s}{\mu} + \frac{2i(s-1)}{\mu(\mu-1)}. \tag{2.2}$$

$$P_{exp-rank}(i) = \frac{1-e^{-1}}{c}. \tag{2.3}$$

The tournament selection is the most commonly used method in GAs, due to its simplicity and efficiency, especially in large populations. The idea of this approach is to avoid considering the entire population for selection by randomly selecting small groups of candidate solutions, comparing them, and then selecting the best one. The pseudocode for this a method is described in Algorithm 2.

Algorithm 2: Pseudocode for the TS algorithm..

```
  Input: $\lambda$, $k$, Population of candidate solutions
  Output: $\lambda$ selected candidate solutions in the mating pool
1 SET current_member to 1; while current_member $\leq \lambda$ do
2     PICK $k$ candidate solutions randomly (with or without replacement)
3     COMPARE the selected candidate solutions and select the best one
4     ADD a copy of the selected candidate solution to the mating pool
5     INCREMENT current_member
6 return $\lambda$ selected candidate solutions in the mating pool
```

The survival selection method, also known as replacement mechanism, is responsible for selecting μ candidate solutions from a set of μ parents plus λ children generated from the μ parents. The aforementioned parent selection mechanisms can be applied to the survival selection. However, some selection mechanisms were specially designed for survival selection and some of them are widely used in the literature. One of the most commonly used mechanism, which is actually complementary to other methods, is elitism. Implementing the elitism mechanism, the current k best candidate solutions are guaranteed to pass to the next generation. Elitism has been found as one of the most important mechanisms to be implemented in any Evolutionary Algorithm, since it greatly influences the convergence of the algorithm, bringing more stability to the search process (EIBEN; SMITH, 2015).

2.2.1.2 Differential Evolution

Differential Evolution (DE) was proposed in 1997 by Storn and Price (STORN; PRICE, 1997) and has been one of the most successful metaheuristics ever created due to its many powerful improved versions proposed over the last two decades, *e.g.* (ZHANG; SANDERSON, 2009; TANABE; FUKUNAGA, 2013; TANABE; FUKUNAGA, 2014; BREST; MAUčEC; BOšKOVIć, 2016; BREST; MAUčEC; BOšKOVIć, 2017; TONG; DONG; JING, 2018). This algorithm was conceived to solve mathematical continuous optimization problems. The main difference between the canonical DE algorithm and GA is the mutation operator called differential mutation. In order to mutate the population, a perturbation vector is calculated for each candidate solution. Such a vector is calculated by randomly choosing three candidate solutions with indexes r_1, r_2 and r_3 and using the formula presented in Equation 2.4, where $F > 0$ is a real number and sets the pace of the search process, and $\mathbf{x_{r_j}}$ is the position of the r_j-th candidate solution (EIBEN; SMITH, 2015).

$$\mathbf{p} = \mathbf{x_{r_1}} + F \cdot (\mathbf{x_{r_2}} - \mathbf{x_{r_3}}). \tag{2.4}$$

A new solution (*i.e.* offspring) is created by exchanging the genes between the perturbation vector and its corresponding parent. The genes from the perturbation vector are inherited by the new solution with a given probability C_r. For each gene not chosen to be passed to the child, the corresponding gene from the selected parent is copied instead. The selection of the surviving solution is made by choosing the fittest one between the parent and the new solution. The general structure of DE algorithms is the same as the GA algorithms (EIBEN; SMITH, 2015).

This canonical strategy is called DE/rand/1/bin. There are plenty of different strategies available in the literature. However, besides the previously described one, another commonly used strategy is the so-called DE/best/1/bin, where instead of using a randomly chosen candidate solution r_1, the best candidate solution (*i.e.* the fittest one) is used, as described in Equation 2.5, where $\mathbf{x_{best}}$ is the position of the best candidate solution in the population (EIBEN; SMITH, 2015).

$$\mathbf{p} = \mathbf{x_{best}} + F \cdot (\mathbf{x_{r_2}} - \mathbf{x_{r_3}}). \tag{2.5}$$

2.2.2 Swarm-based Algorithms

Swarm-based algorithms, also known as Swarm Intelligence (SI) algorithms, are powerful metaheuristics based on a population of candidate solutions that searches in parallel for the optimal solution(s) of an objective function in a search space. The communication between such very simple reactive agents causes the emergence of complex self-organizing patterns capable of efficiently solving very complex optimization problems. This emergence property is called Swarm Intelligence, hence the name of the field (BONABEAU; DORIGO; THERAULAZ, 1999; PANIGRAHI; SHI; LIM, 2011; VASUKI, 2020). These mechanisms are inspired by the behavior of collectives of animals in nature, *e.g.* flocks of birds looking for good sources of food or schools of fish protecting themselves from predators. In general, the candidate solutions in the best spots in the search space share their positions with the others, so that each candidate solution decides how such an information will be used together with the knowledge acquired throughout its own search path. The stronger the influence of the best-positioned candidate solutions on the population, the stronger the exploitation behavior of the swarm. The stronger the influence on each candidate solution of its own search history, the stronger the exploration behavior of the population. As well as EAs, the biggest challenge in swarm-based algorithms is to manage an adequate balance between exploration and exploitation throughout the search process.

Three swarm-based algorithms are detailed below: Particle Swarm Optimization (PSO), Ant Colony Optimization (ACO) and Fish School Search (FSS). These algorithms are chosen to be detailed in this section because they were used in the experimental benchmark of this work. Their selection was mainly because of their distint operational characteristics, respectively, combined optimization rationale on local and global information, ample schem of externally shared information, and automatic selection of exploration and exploitation.

2.2.2.1 Particle Swarm Optimization

The Particle Swarm Optimization (PSO) algorithm was proposed by Kennedy and Eberhart in 1995 (KENNEDY; EBERHART, 1995). It is inspired on the collective foraging behavior of birds and its one of the most widely used and well-succeeded swarm-based algorithm so far. Its simple yet effective mechanism is described in Algorithm 3. In this pseudocode, it can be seen that, for each iteration, the velocity vector of each candidate solution is calculated, then it is added to the position vector. It is important to notice that it was originally conceived to continuous search spaces. If the new position of a given particle is outside the boundaries of the search space, the position vector is not updated. The equation of the velocity vector defined in the Equation 2.6 was proposed in 1998 by Shi and Eberhart (SHI; EBERHART, 1998) and it is the most commonly used version of the technique. It adds the parameter w (*i.e.* inertia weight) to the canonical algorithm. In this Equation, $\mathbf{v_{i,t}}$ is the velocity vector in $\mathbb{R}^n$ of the candidate solution i at iteration t, c_1 and c_2 are real-valued parameters called cognitive and social factors, respectively, $\mathbf{r_{i_1}}$ and $\mathbf{r_{i_2}}$ are two random vectors in $[0,1]^n$, $\mathbf{x_{i,t}}$ is the position vector in $\mathbb{R}^n$ of the candidate solution i at iteration t, $\mathbf{p_{i,t}}$ is the personal best (pbest) vector in $\mathbb{R}^n$ of the candidate solution i at iteration t, and $\mathbf{g_t}$ is the global best (gbest) vector in $\mathbb{R}^n$ of the entire population at time t.

Algorithm 3: Pseudocode for the PSO algorithm.

```
   Input: Randomly initialized population of candidate solutions
   Output: Best solution ever found
1  EVALUATE each candidate solution
2  while Stopping condition is not met do
3      CALCULATE velocity vector for each candidate solution according to
         Equation 2.6
4      ADD the velocity vectors of each candidate solution to their position vectors
         according to Equation 2.7
5      EVALUATE each candidate solution
6      UPDATE the candidate solutions' pbest position and the gbest position
7  return Best solution ever found
```

$$\mathbf{v_{i,t+1}} = w * \mathbf{v_{i,t}} + c_1 * \mathbf{r_{i_1,t}} \cdot (\mathbf{p_{i,t}} - \mathbf{x_{i,t}}) + c_2 * \mathbf{r_{i_2,t}} \cdot (\mathbf{g_t} - \mathbf{x_{i,t}}). \tag{2.6}$$

$$\mathbf{x_{i,t+1}} = \mathbf{x_{i,t}} + \mathbf{v_{i,t+1}}. \tag{2.7}$$

Each dimension in the velocity vector is limited to a maximum value in order to avoid instability in the population. The pbest of the candidate solution i at iteration t is the best position found by the candidate solution itself until the beginning of the iteration t, while the gbest of the population at iteration t is the best position found by the entire population until the beginning of the iteration t. The inertia weight w is the weight given to the previous velocity vector, which conserves part of its direction and intensity. The higher its value, the stronger the exploration behavior. The cognitive factor c_1 is the weight given to the difference vector between the pbest and the current position of each candidate solution, *i.e.* the weight given to its own knowledge acquired throughout the search process. The higher its value, the stronger the exploration behavior in the algorithm. Finally, the social factor c_2 is the weight given to the difference vector between the gbest and the current positions of the candidate solutions, *i.e.* the weight given to the the global knowledge acquired throughout the search process. Thus, the higher its value, the stronger the exploitation behavior on the population.

Despite the enormous success of such an algorithm in many applications, it usually presents an overly exploitative behavior. As well as other metaheuristics, the maintenance of the balance between exploration and exploitation has been one of its greatest challenges. Thus, the PSO algorithm and its many variants have been widely studied since the proposal of the canonical version of the metaheuristic. In 2006, Liang *et al.* have proposed an interesting version of the PSO called Comprehensive Learning PSO (CLPSO) (LIANG et al., 2006). In their study, they have proposed a modification to the velocity vector as shown in Equation 2.8, where $\mathbf{p^*_{i,t}}$ is a vector of the candidate solution i updated according to Equations 2.9 and 2.10. Such an update occurs every m iterations with no improvement in the fitness of the given particle. In Equation 2.9, $p^*_{i_j,t+1}$ is the j^{th} dimension of the vector $\mathbf{p^*_i}$ at the iteration $t+1$, $p_{i_j,t}$ is the j^{th} dimension of the pbest vector of the candidate solution i at the iteration t, $p_{TS_j,t}$ is the j^{th} dimension of the pbest vector of another candidate solution selected by tournament at the iteration t, $r_{i_j,t}$ is a random number between 0 and 1, and ps is the population size. In the CLPSO original paper, $a = 0.05$ and $b = 0.45$.

$$\mathbf{v_{i,t+1}} = w * \mathbf{v_{i,t}} + c * \mathbf{r_{i_1,t}} \cdot (\mathbf{p^*_{i,t}} - \mathbf{x_{i,t}}). \tag{2.8}$$

$$p^*_{i_j,t+1} = \begin{cases} p_{i_j,t}, & \text{if } r_{i_j,t} \leq p_{C_i,t}, \\ p_{TS_j,t}, & otherwise. \end{cases} \tag{2.9}$$

$$p_{C_i,t} = a + b * \frac{(exp((10(i-1))/(ps-1)) - 1)}{(exp(10) - 1)}. \tag{2.10}$$

In the proposed mechanism, each dimension of the $\mathbf{p^*_i}$ vector is updated separately. For each dimension, if a random number is lower than the probability calculated according to Equation 2.10, the corresponding value of the pbest of the candidate solution i is copied into the vector. Otherwise, the corresponding value of the pbest of another candidate solution chosen by tournament selection between two randomly chosen candidate solutions in the remaining population is copied instead. According to Equation 2.10, the probability of choosing its own pbest to update the vector $\mathbf{p^*_i}$ by the candidate solution

i depends on its index. The higher the probability, the more exploratory is its behavior. On the other hand, the lower the probability, the more exploitative it is. It means that the balance between exploration and exploitation varies among the candidate solutions in the population according to their indexes.

Even though candidate solutionizing the levels of exploration and exploitation is an important improvement on the PSO algorithm, candidate solutions with higher exploration tendency are adversely influenced by candidate solutions with a higher exploitation tendency. Such an issue was addressed by Lynn and Suganthan in 2015, when they proposed one of the current state-of-the-art PSO variants, the Heterogeneous Comprehensive Learning PSO (HCLPSO) (LYNN; SUGANTHAN, 2015). In this algorithm, the population is divided into two subpopulations: an exploratory and an exploitative subpopulation. The velocity of the particles in the exploratory subpopulation is calculated through Equation 2.8, while the candidate solutions in the exploitation subpopulation calculate their velocities according to Equation 2.11. The authors of this paper have defined that $a = 0$ and $b = 0.25$. In HCLPSO, the vector $\mathbf{p^*_{i,t}}$ for the candidate solutions of the exploratory subpopulation is updated considering only the candidate solutions from the same subpopulation, while the candidate solutions from the exploitation subpopulation learn from the entire population. Besides, if the pbest vector of an candidate solution i from the exploration subpopulation is copied into ($\mathbf{p^*_{i,t}}$ when this vector is updated, a dimension is randomly chosen to be learnt from the pbest of a randomly chosen candidate solution in the same subpopulation.

$$\mathbf{v_{i,t+1}} = w * \mathbf{v_{i,t}} + c_1 * \mathbf{r_{i_1,t}} \cdot (\mathbf{p^*_{i,t}} - \mathbf{x_{i,t}}) + c_2 * \mathbf{r_{i_2,t}} \cdot (\mathbf{g_t} - \mathbf{x_{i,t}}). \tag{2.11}$$

With the aforementioned mechanisms, the authors have proposed a more powerful PSO variant than CLPSO, with a more effective balance between exploration and exploitation. The results achieved in the experiments presented in the paper still place the proposed algorithm among the state-of-the-art variants of PSO.

2.2.2.2 Ant Colony Optimization

The vanilla version of the Ant Colony Optimization algorithm was proposed in 1992 by Dorigo (DORIGO, 1992). Conceived to solve combinatorial optimization problems, it is inspired on the foraging behavior of ants. In this algorithm, the problem is modelled as a graph where the nodes are the possible values of the solution space. The objective of these ants is to find the less expensive path where all nodes are visited only once. Every candidate solution must walk through the graph looking for such a path and calculating the cost of each valid path found. Such a cost then defines the amount of pheromone deposited by each candidate solution in the corresponding path. The less expensive is the path, the more pheromone will be deposited on the links between the nodes that formed the currently evaluated path. Thus, the best paths are strongly marked by large amounts of pheromone. For each new iteration, the more pheromone deposited in a given path, the more likely its choice. It is easy to see that the application of such an algorithm to the Traveling Salesman Problem (TSP) is straightforward (ILAVARASI; JOSEPH, 2014). The very simple pseudocode for the ACO algorithm is depicted in Algorithm 4.

Algorithm 4: Pseudocode for the vanilla ACO algorithm.

```
  Input: Graph with initial values of pheromone
  Output: Best path ever found
1 while Stopping condition is not met do
2     CONSTRUCT the ant solutions
3     UPDATE globally the pheromone trails
4 return Best solution ever found
```

The amount of pheromone τ_0 deposited on each link of the graph must be previously defined by the user. The construction of a new solution is made by each ant starting from an initial node and gradually choosing the next step (*i.e.* next solution to add to the final solution vector). For each visited node i, the next node j is chosen probabilistically by looking at the available nodes connected to i. Equation 2.12 shows the most widely used formula to calculate the probability of choosing a node j from a node i, where c_{ij} represents the link between the nodes i and j, $S_{p,t}$ is the set of available nodes given the partial solution build so far by the ant, $\tau_{ij,t}$ is the amount of accumulated pheromone deposited in the link between the nodes i and j up to the current iteration t, $\eta(c_{ij})$ is the cost of walking from node i to node j, and α and β are parameters that defines the importance of the knowledge acquired by the population during the search process and the *a priori* knowledge defined by the user, respectively (DORIGO; MANIEZZO; COLORNI, 1996).

$$p(c_{ij}|S_{p,t}) = \frac{\tau_{ij,t}^{\alpha}[\eta(c_{ij})]^{\beta}}{\sum_{c_{il}\in S_{p,t}} \tau_{il,t}^{\alpha}[\eta(c_{il})]^{\beta}}. \tag{2.12}$$

The pheromone update in a given link between two nodes i and j is usually made through Equation 2.13, where ρ is the evaporation rate, $g(\mathbf{s})$ is the quality (*i.e.* fitness) assigned to the solution solution vector $\mathbf{s}$, and $S_{upd,t}$ is the set of solutions selected to update the pheromone trails.

$$\tau_{ij,t+1} = (1-\rho)\tau_{ij,t} + \sum_{\mathbf{s}\in S_{upd,t}} g(\mathbf{s}). \tag{2.13}$$

In 1997, Stutzle and Hoos proposed a widely used variant of the ACO algorithm, the MAX-MIN Ant System (MMAS) (STUTZLE; HOOS, 1997). In this study, the authors proposed maximum and minimum bounds for the accumulated amount of pheromone deposited in the links. Moreover, the S_{upd} set includes only one solution: the best solution in the current iteration. However, from time to time the best solution ever is used instead.

Such a frequency is increased over time, increasing the exploitation of the search process as it comes closer to the end of the optimization process. Since MMAS was applied to the TSP problem, the quality of the solutions are their corresponding travelling distances $d(\mathbf{s})$. Since the objective of the TSP is to minimize the travelling distance, $g(\mathbf{s})$ is defined as $1/d(\mathbf{s})$.

2.2.2.3 Fish School Search

Proposed by Bastos Filho and Lima Neto in 2008, Fish School Search (FSS) (FILHO et al., 2008)(FILHO et al., 2009a) is a PBA for continuous optimization problems. It is inspired on the collective foraging behavior of fish schools. The success of the search process is represented by the weight of each fish. In other words, the heavier an candidate solution, the more successful its search history.

Algorithm 5 shows the pseudocode of the FSS algorithm. Each iteration of the algorithm is divided into four operators: the candidate solution movement, the feed operator, the collective instinctive movement, and the collective volitive movement. It is important to mention that two solution evaluations are called for each iteration. This is necessary for the feeding operator, as explained later in this subsection.

Algorithm 5: Pseudocode for the FSS algorithm.

```
  Input: Randomly initialized population of candidate solutions
  Output: Best solution ever found
1 EVALUATE each candidate solution
2 while Stopping condition is not met do
3     Run the candidate solution MOVEMENT operator
4     EVALUATE each candidate solution
5     Run the FEEDING operator
6     Run the COLLECTIVE INSTINCTIVE MOVEMENT operator
7     Run the COLLECTIVE VOLITIVE MOVEMENT operator
8     EVALUATE each candidate solution
9 return Best solution ever found
```

In the candidate solution movement operator, each candidate solution randomly moves in the search space. Such an operator is responsible for the exploratory behavior of the algorithm. In this operator, random vectors are added to each solution in the population, as shown in Equation 2.14, where $step_{ind}$ is the candidate solution step and must be defined by the user. After executing such an operator for each fish, the candidate solution only moves to the new position if it is better than the current one.

$$\mathbf{x_{i,t+1}} = \mathbf{x_{i,t}} + \mathbf{r_{i,t}} * step_{ind}. \tag{2.14}$$

As already mentioned, the success of each fish is encoded in the weight of the candidate solution. The fish gains weight through the feeding operator, where the relative fitness gain after the last execution of the candidate solution movement is added to the previous weight. Such a mechanism is shown in Equation 2.15, where $\Delta f_{i,t}$ is the fitness gain of the candidate solution i after the candidate solution movement at the current iteration t, and $\mathbf{w_{i,t}}$ is the weight of the i-th fish at iteration t.

$$w_{i,t+1} = w_{i,t} + \frac{\Delta f_{i,t}}{\max_i \Delta f_{i,t}}. \tag{2.15}$$

The collective instinctive movement defines a displacement vector that is added to the position vector of all candidate solutions in the population. Such a vector is the weighted average of the displacement vectors of each candidate solution. This displacement vector is the difference vector between the positions of a given candidate solution after and before the candidate solution movement. The weights for this average calculation are the fitnesses gained during the candidate solution movement for each candidate solution. Equation 2.16 shows the calculation of a new position vector in the collective instinctive movement, where N is the population size and $\Delta\mathbf{x_{k,t}}$ is the aforementioned displacement vector added to the candidate solution k during the displacement movement at the current iteration t.

$$\mathbf{x_{i,t+1}} = \mathbf{x_{i,t}} + \frac{\sum_{k=1}^{N} \Delta f_{k,t} * \Delta\mathbf{x_{k,t}}}{\sum_{k=1}^{N} \Delta f_{k,t}}. \tag{2.16}$$

The collective volitive movement is the last operator executed in every iteration of the FSS. It is responsible to increase the exploitation or the exploration according to the current situation of the search process. The exploitation is increased by contracting the fish school when the total weight sum of the population increases between the latest and the current iterations. On the other hand, the exploration is increased by expanding the population when the total weight remains the same. It is important to highlight that the weight of a fish never decreases, since a fish updates its position on the candidate solution movement only if its fitness increases. Thus, the gain of weight of any fish can be only positive or null. Surely, the same applies to the total weight of the population. The total weight stagnation happens when the algorithm gets stuck in a local minimum and cannot find better places during the candidate solution movement. When this happens, the algorithm expands its population, increasing the exploration behavior, what makes the fish visit areas that have not been visited before (most likely).

The movements of expansion and contraction are performed using the barycenter of the population as a reference point. The barycenter is calculated through Equation 2.17. The expansion movement is described in the Equation 2.18 and the contraction movement is shown in the Equation 2.19. $step_{vol}$ is the volitive step size that must be defined by the user.

$$\mathbf{B_t} = \frac{\sum_{i=1}^{N} \Delta w_{i,t} * \Delta\mathbf{x_{i,t}}}{\sum_{i=1}^{N} \Delta w_{i,t}}. \tag{2.17}$$

$$\mathbf{x_{i,t+1}} = \mathbf{x_{i,t}} + step_{vol} r_{i,t} \frac{\mathbf{x_{i,t}} - \mathbf{B_t}}{||\mathbf{x_{i,t}} - \mathbf{B_t}||}. \tag{2.18}$$

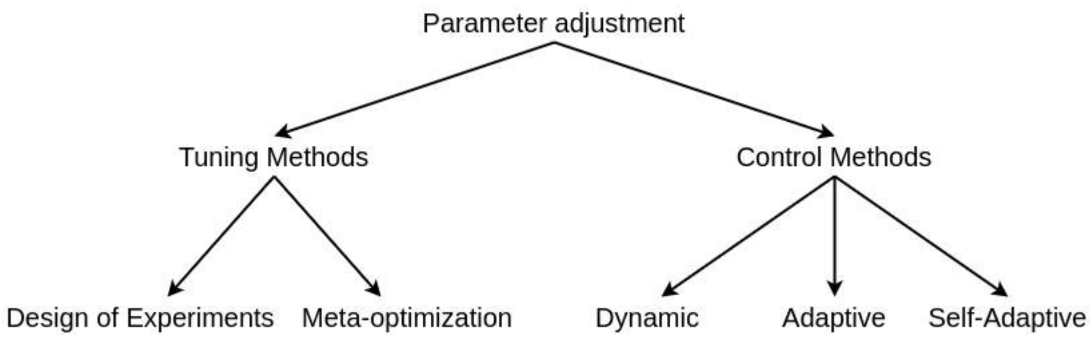
$$\mathbf{x_{i,t+1}} = \mathbf{x_{i,t}} - step_{vol} r_{i,t} \frac{\mathbf{x_{i,t}} - \mathbf{B_t}}{||\mathbf{x_{i,t}} - \mathbf{B_t}||}. \tag{2.19}$$

2.2.3 Parameter Adjustment for Metaheuristics

As already mentioned, according to the No Free Lunch Theorem, there is no parameter setting for a given algorithm that works best for every possible problem (WOLPERT; MACREADY, 1997). Therefore, finding an optimal parameter setting that maximizes the performance of a given metaheuristic in a specific problem is necessary. However, manual parameter adjustment for EA and SI can be very hard and time demanding. Therefore, automating this task has been one of the greatest and most important challenges in the field (EIBEN; HINTERDING; MICHALEWICZ, 1999).

Autonomous adjustment approaches can be divided into two groups (KARAFOTIAS; HOOGENDOORN; EIBEN, 2015b): tuning algorithms and control algorithms. Figure 5 shows the taxonomy of the methods of parameter adjustment. Such a classification was adapted from (TALBI, 2009) and (EIBEN; SMITH, 2015). The following subsections depicts each one of these classes.

Figure 5 – Taxonomy of parameter adjustment techniques.



Source: Produced by the author based on (TALBI, 2009) and (EIBEN; SMITH, 2015).

2.2.3.1 Parameter Tuning

Parameter tuning methods, also known as off-line methods, are techniques that automatically find good parameters for a given metaheuristic previous to its execution on a target problem, keeping them constant throughout its execution. Usually, metaheuristic designers or users tune one parameter at a time. Thus, the interferences between some of the parameters are not captured and the results may be suboptimal. To overcome such an issue, Design of Experiments (DOE) can be used. In order to use DOE, a few things must be defined beforehand (TALBI, 2009):

- Parameters to vary in the experiments (also known as factors, design variables, predictor variables, and input variables);
- Levels that represent the possible values of each parameter to be adjusted.

When n parameters need to be adjusted with k levels each one, brute-force approaches, such as Grid Search, need k^n experiments to find a good adjustment (LIASHCHYNSKYI; LIASHCHYNSKYI, 2019). It is important to notice that each experiment is composed by a number of executions of the metaheuristic with a given parameter setup for a set of training functions. However, there are more efficient methods for experimental design, where a smaller number of experiments is created. For example, the F-Race algorithm initially creates a set of experiments and runs each one of them on a set of training problems (BIRATTARI et al., 2010). Then, instead of running all setups for all training functions, the experiments are gradually evaluated on such a problem set and the setups that show poor performance can be earlier discarded, what saves processing time (BIRATTARI et al., 2002).

An iterated version of the F-Race tuning algorithm, known as Iterated F-Race (I/F-Race), starts with a small set of setups, applies the F-Race algorithm until a given stopping condition is met. Then, it randomly generates new setups using the distribution of the surviving setups. Such a technique has been widely used since its proposal and it can be considered as one of the state-of-the-art techniques of parameter tuning for metaheuristics (BALAPRAKASH; BIRATTARI; STÜTZLE, 2007a; BIRATTARI et al., 2010).

As depicted in Figure 5, tuning methods can also be classified as meta-optimization algorithms. The techniques in this group apply a metaheuristic to search for a good parameter setup of another metaheuristic. In such approaches, the solution of the upper metaheuristic is the parameter setup of the lower metaheuristic. Thus, a solution evaluation consists in an execution (or multiple executions for non-deterministic metaheuristics) of the lower metaheuristic with the parameter setup encoded in the upper solution vector (BIRATTARI et al., 2002).

2.2.3.2 Parameter Control

Tuning methods are widely used in EA and SI communities, especially due to their usually simple mechanisms. However, it is well-known that the optimal value of a given parameter changes along the optimization problem. Parameter control algorithms, also known as online parameter adjustment algorithms, set the parameter values of a metaheuristic on-the-fly, *i.e.* throughout the optimization process. Thus, they are capable of choosing the best parameter setup for a given algorithm solving a given problem at a given point in time. Therefore, a metaheuristic with dynamic parameters are likely to perform better than the same metaheuristic with static parameters (EIBEN; HINTERDING; MICHALEWICZ, 1999; KARAFOTIAS; HOOGENDOORN; EIBEN, 2015b).

As shown in Figure 5, there are three types of parameter control algorithms: dynamic, adaptive, and self-adaptive methods (EIBEN; SMITH, 2015). The dynamic methods do not use any information from the search process in order to make decisions about the parameters values. These decisions are usually made based on the elapsed time of the search process (EIBEN; SMITH, 2015). For example, a common dynamic parameter control policy for the PSO algorithm is a linear decrease of the inertia weight and the cognitive factor, and a linear increase of the social factor. Such policies are usually conceived in order to create a stronger exploratory behavior in the beginning of the search process followed by a stronger exploitative behavior in the end.

The adaptive methods receive feedbacks from the search process so that smarter decisions can be made. Adaptive techniques involve mechanisms of credit assignment, which estimates the importance of each decision on the success of the entire search process. It is important to mention that the control mechanism is implemented in an external agent that interacts with the running metaheuristic, setting parameter values and receiving feedbacks on its performance with the new parameter adjustment (EIBEN; SMITH, 2015).

Finally, the self-adaptive methods evolve the parameters of a metaheuristic by including them in the solution vector. Thus, the parameters will be evolved alongside the decision variables of the optimization problem by the algorithm itself. It means that even though self-adaptive methods also use the feedback of the search process to make decisions, the parameter control policy is implicitly implemented with the operators of the metaheuristic itself. For example, in a self-adaptive approach for GA with real-valued solution vector, the mutation and the crossover probabilities are encoded in the solution vector itself and are treated as decision variables.

As already mentioned in this work, a different classification scheme was proposed in (KARAFOTIAS; HOOGENDOORN; EIBEN, 2015b): control methods tailored to an application and out-of-the-box control methods. The parameter control algorithms tailored to specific applications are techniques that were designed to work with a specific metaheuristic and/or a specific problem. The vast majority of the studies about parameter control methods for EA and SI that have been published so far focus on such methods (LACERDA et al., 2021). Such a finding is better discussed in the literature review chapter of this study. On the other hand, out-of-the-box parameter control algorithms can be applied to several metaheuristics and optimization problems. It means that without any change in the logic of the control algorithm, it can be used in different scenarios. The differences between both groups are similar to the differences between customized and out-of-the-box softwares. It is important to note that out-of-the-box methods are inherently adaptive, since in order to be able to be applied to multiple metaheuristics and optimization problems, the control mechanism must be implemented outside the metaheuristic itself. Therefore, no self-adaptive methods should be classified as out-of-the-box.

This work focuses on out-of-the-box methods. Therefore, in chapter 3, a literature review on this subject is presented.

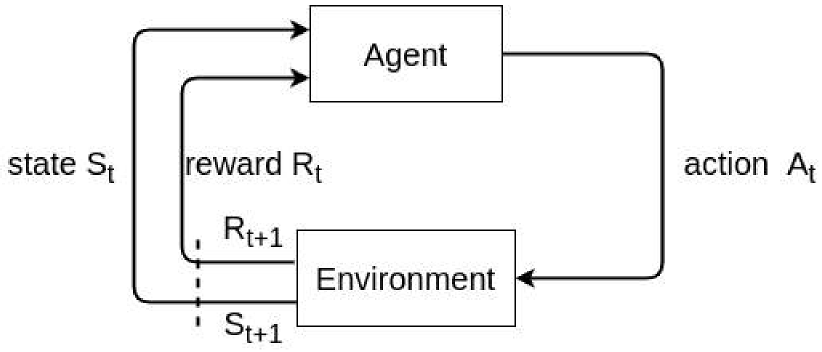
2.3 REINFORCEMENT LEARNING

Reinforcement Learning (RL) is a machine learning paradigm where an agent interacts with its surrounding environment by taking actions and receiving rewards, improving its performance by trial-and-error. This section describes the basics of RL and presents the necessary topics to understand the proposal of this work.

An RL problem can be formulated as a Markov Decision Problem (MDP), which is a classical formal formulation of sequential decision making processes. In an MDP, the decision maker (*i.e.* learner) is called *agent*. Everything outside the agent is called *environment*, which the agent interacts with through *actions* that modify its state. Since the agent is goal-oriented, which means that there must be an objective that the agent must pursue, the effect of each action on the environment generates a reward that indicates how good is the taken action for the pursuit of the agent's objective. A positive reward reinforces the taken action for the observed state when the given action was taken (SUTTON; BARTO, 2018).

As already mentioned, in an RL problem the learning agent interacts with the environment over and over through a trial-and-error process. In this iterative process, it is expected that the agent learns optimum actions for different scenarios (*i.e.* states of the environment). Figure 6 shows this mechanism, where the agent takes and action A_t at time t, and the environment "reacts" to the action, returning a new state S_{t+1} alongside a reward R_{t+1}. The sequence of states, actions, and rewards $S_0, A_0, R_1, S_1, A_1, R_2, S_2, A_2, R_3$... is called *trajectory* (SUTTON; BARTO, 2018).

Figure 6 – The agent-environment interaction in a Markov decision process.



Source: Produced by the author based on (SUTTON; BARTO, 2018).

Equation 2.20 shows the probability of reaching a state s' at time t and receive a reward r after taking an action a at time t, when the environment was in a previous state S_{t-1} at time $t-1$. In such a formulation, R_t and S_t are random variables whose distributions are dependent only on the preceding state and action. Such a definition defines the dynamics of the MDP (SUTTON; BARTO, 2018).

$$p(s', r|s, a) = P_r\{S_t = s', R_t = r|S_{t-1} = s, A_{t-1} = a\}. \tag{2.20}$$

The objective of an RL agent is to maximize the cumulative reward received along the trajectory. Thus, the reward can be thought of as a scalar signal of which accumulation over time must be maximized. The accumulated reward is also known as *expected return* (PUTERMAN, 1990).

A *terminal state* of an MDP is a state that, whenever the agent reaches it, the agent stops learning. A full trajectory from an initial state to a terminal state is called *episode*. Learning tasks with such a state are called *episodic tasks*, of which the expected return can be calculated according to Equation 2.21. However, in many situations (especially in robotics), there is no such a state, which means that the agent must learn indefinitely. These tasks are called *continuing tasks*. In both cases, the agent tries to maximize G_t for each time t (PUTERMAN, 1990; SUTTON; BARTO, 2018).

$$G_t = R_{t+1} + R_{t+2} + R_{t+3} + ... + R_T. \tag{2.21}$$

The problem of continuing tasks is that the expected return is usually infinite. In order to overcome this issue, the agent must consider a decreasing discounting factor for each of the future reward as described in Equation 2.22. In this equation, $\gamma \in [0, 1]$ and is called *discounting factor* (KAELBLING; LITTMAN; MOORE, 1996; SUTTON; BARTO, 2018). It is important to note that the closer to zero, the more myopic is the agent. On the other hand, the closer to one, more importance is given to distant rewards. G_t can also be calculated recursively, as defined in Equation 2.23.

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + ... = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}. \tag{2.22}$$

$$G_t = R_{t+1} + \gamma G_{t+1}. \tag{2.23}$$

In order to unify the representation of both episodic and continuing tasks, instead of using terminal states, *absorbing states* can be added to the MDP, from which transitions take the execution flow back to themselves and the corresponding rewards are zero (ALAGOZ et al., 2010). Then, the expected return can be defined according to Equation 2.24, where T can be infinite or not (SUTTON; BARTO, 2018).

$$G_t = \sum_{k=t+1}^{T} \gamma^{k-t-1} R_k. \tag{2.24}$$

Almost every RL algorithm works by iteratively learning a function that estimates how good is a given state, or a given action taken in a given state. Such an estimation is intended to make the agent choose the best action for the current state in order to maximize the expected return. The probability function that maps each state to each possible action is called *policy*, which can be represented as a function $\pi(a|s)$, where a is the taken action and s is the current state of the environment (SUTTON; BARTO, 2018).

Given an agent with a policy π, the value function of a state s is the expected return if the agent starts working in the given state and follows π thereafter. In other words, the quality of a given state is the expected return of an agent using a policy π, starting from the given state until it reaches an absorbing state. Equation 2.25 presents such a function under the MDP framework, which is called *state-value function for policy* π. Similarly, the *action-value function for policy* π can be computed as shown in Equation 2.26, which computes the expected return (*i.e.* quality) of an action a taken when the agent observes the environment in a state s. As already mentioned, the value functions v_π and q_π can be learned from experience, when true rewards are received for each state and taken action (LAPAN, 2018).

$$v_\pi(s) = \mathop{\mathbb{E}}_{\pi}[G_t|S_t = s] = \mathop{\mathbb{E}}_{\pi}\left[\sum_{k=0}^{\infty}\gamma^k R_{t+k+1}|S_t = s\right], \forall s \in S. \tag{2.25}$$

$$q_\pi(s,a) = \mathop{\mathbb{E}}_{\pi}[G_t|S_t = s, A_t = a] = \mathop{\mathbb{E}}_{\pi}\left[\sum_{k=0}^{\infty}\gamma^k R_{t+k+1}|S_t = s, A_t = a\right]. \tag{2.26}$$

Equation 2.27 shows the relationship between the value function of a state s and the state values of its successor states. This is called *Bellman equation*. It computes recursively (*i.e.* as a breadth-first search in a tree of possibilities) the expected return by considering all possibilities of states, actions, and rewards from the current state until it reaches an absorbing one. In other words, for each combination of a, s' and r, its probability is calculated as $\pi(a|s)p(s',r|s,a)$. Then, the expected return is calculated by summing over the rewards of all visited states multiplied by their probabilities (MA; STACHURSKI, 2019).

$$\begin{aligned} v_\pi(s) &= \mathop{\mathbb{E}}_{\pi}[G_t|S_t = s] \\ &= \mathop{\mathbb{E}}_{\pi}[R_{t+1} + \gamma G_{t+1}|S_t = s] \\ &= \sum_a \pi(a|s)\sum_{s'}\sum_r p(s',r|s,a)\left[r + \gamma \mathop{\mathbb{E}}_{\pi}[G_{t+1}|S_{t+1} = s']\right] \\ &= \sum_a \pi(a|s)\sum_{s',r} p(s',r|s,a)\left[r + \gamma v_\pi(s')\right], \forall s \in S. \end{aligned} \tag{2.27}$$

Given the definitions presented so far, the goal of an RL agent can be redefined as the search for a policy that maximizes the state and state-action value functions, *i.e.* maximizes the expected return for each visited state in its trajectory. A policy π is better or equal than a policy π' if and only if $v_\pi(s) \geq v_{\pi'}(s)$, $\forall s \in S$. There is always a policy that is better than or equal to all other policies. Such a policy is called *optimal policy* and can be denoted as π_*. Optimal policies make optimal decisions, what causes the state value function to be optimal as well (LAPAN, 2018). Equation 2.28 shows the definition of the optimal state-action value function, where $v_*(S_{t+1})$ is the optimal state value function, as defined in Equation 2.29.

$$q_{\pi_*}(s,a) = \mathbb{E}[R_{t+1} + \gamma v_*(S_{t+1})|S_t = s, A_t = a]. \tag{2.28}$$

$$v_{\pi_*}(S_{t+1}) = \max_{\pi} v_\pi(s). \tag{2.29}$$

Since v_* is the value function of a policy, it must satisfy the self-consistency posed by the Bellman equation. Equation 2.30 shows the Bellman equation for v_*, also known as *Bellman optimality equation*, where $A(s)$ is the set of available actions for the state s. Such an equation is built upon the idea that optimal actions are always taken with optimal policies, as previously mentioned (**??**LAPAN, 2018).

$$\begin{aligned} v_*(s) &= \max_{a \in A(s)} q_{\pi_*}(s,a) \\ &= \max_a \mathop{\mathbb{E}}_{\pi_*}[G_t|S_t = s, A_t = a] \\ &= \max_a \mathop{\mathbb{E}}_{\pi_*}[R_{t+1} + \gamma G_{t+1}|S_t = s, A_t = a] \\ &= \max_a \mathbb{E}[R_{t+1} + \gamma v_{\pi_*}(S_{t+1})|S_t = s, A_t = a] \\ &= \max_a \sum_{s',r} p(s',r|s,a)[r + \gamma v_{\pi_*}(s')]. \end{aligned} \tag{2.30}$$

The Bellman optimality equation can be used to compute the state-action value function for the optimal policy, as defined in Equation 2.31.

$$q_{\pi_*}(s,a) = \sum_{s',r} p(s',r|s,a) \left[r + \gamma \max_{a'} q_{\pi_*}(s',a') \right]. \tag{2.31}$$

The Bellman optimality equation can be exactly solved. However, such an exact solution can only be found under three conditions: the dynamics of the environment is accurately known; there is enough computational resource to compute the exact solution in a reasonable time; the RL problem presents the Markov property. *Dynamic Programming* (DP) is a set of widely known iterative algorithms that guarantee optimal policies in MDPs where their dynamics are perfectly known. They are quite efficient under such conditions when compared to other methods such as linear programming or direct search. Two of the most common DP approaches are Policy Iteration and Value Iteration (BERTSEKAS, 2000; LAPAN, 2018).

In the Policy Iteration method, two iterative processes are interchangeably executed: policy evaluation and policy improvement. The output of the policy evaluation affects the policy improvement and vice-versa. Such a “cooperation" happens until the the optimal policy is found (LAPAN, 2018). The policy evaluation and policy improvement processes are shown in Equations 2.32 and 2.33, respectively, which are computed for each state $s \in S$.

$$v_{k+1}(s) = \sum_{s',r} p(s', r|s, \pi_k(s))[r + \gamma v_k(s')]. \tag{2.32}$$

$$\pi_{k+1}(s) = \arg\max_a \sum_{s',r} p(s', r|s, \pi_k(s))[r + \gamma v_{k+1}(s')]. \tag{2.33}$$

It is important to mention that these iterative processes are executed until they converge to the optimal policy or another stopping criterium is met. However, defining an effective and efficient criterium can be hard. The Value Iteration method serves as an alternative technique, where a single iteration is performed for both iterative processes. Thus, it can be written as a single iterative process that combines the policy improvement and the policy evaluation steps, as shown in Equation 2.34. Since an optimal state value function always leads to optimal actions, computing such an optimal function takes the search process to the optimal policy (LAPAN, 2018).

$$v_{k+1}(s) = \max_a \sum_{s',r} p(s', r|s, \pi_k(s))[r + \gamma v_k(s')]. \tag{2.34}$$

Although these techniques guarantee the return of the optimal policy, a perfect model of the environment's dynamics is almost impossible to be obtained in real world possible. Therefore, methods that approximate the state and state-action value functions are mandatory for such cases. The following subsections present some of the most common algorithms from this group, which are keys for the success of the method proposed in this work.

2.3.1 Temporal-Difference Learning

Monte Carlo are RL methods that do not rely on the complete knowledge of the environment. Instead, they learn the value functions from a set of *experiences*, *i.e.* sequence of states, actions, and rewards sampled from the interactions between the agent and the environment. In such methods, the adjustment of the value functions is performed after a full episode is executed. In the *every-visit* Monte Carlo method, the state value function is calculated according to Equation 2.35, where G_t is the actual return following time t, and α is the learning rate. In such an equation, G_t serves as a target value to which $v_k(S_t)$ should converge. In this equation, k is the number of the current episode (LAPAN, 2018).

$$v_{k+1}(s_t) = v_k(s_t) + \alpha \left[G_t - v_k(s_t)\right]. \tag{2.35}$$

Waiting until the end of an episode to assign credits to the states may cause the credit assignment process to be inaccurate, since the details of the trajectory are lost when the rewards are summarized. Thus, instead of updating the value function at the end of the episode, Temporal-Difference (TD) methods wait only until the next step. However, the actual cumulative return is not known when such an update is made (APOSTOL, 2012). Therefore, as defined in the Bellman equations in Equation 2.27, the expected return (*i.e.* the target value) is estimated through $r_(t+1)+\gamma v_t(S_{t+1})$ immediately after the transition from s_t to s_{t+1}, when the reward r_{t+1} is returned.

The state value function update of the simplest TD method, known as *TD(0)*, or *one-step* TD, is shown in Equation 2.36. For each step t in the episode, the TD(0) agent perceives the environment in the state s_t, takes an action a_t following a policy π, observes the new state s_{t+1} and the received reward r_{t+1}, and computes the new state value $v_{t+1}(s_t)$. Notice that the quantity between brackets is the error between the estimated state value $v_t(s_t)$ and the estimate $r_(t+1)+\gamma v_t(S_{t+1})$. Such an error is called *TD error* and is usually represented as δ_t, which is guaranteed to converge (*i.e.* stabilize) in the long run. Even though the difference between G_t and $v_k(s_t)$ in the every-visit Monte Carlo method is also guaranteed to converge, the TD error in TD(0) converges faster (TESAURO, 1992; LAPAN, 2018).

$$v_{t+1}(s_t) = v_t(s_t) + \alpha\left[r_{t+1} + \gamma v_t(s_{t+1}) - v_t(s_t)\right]. \tag{2.36}$$

In the description of the TD(0) method presented in the previous paragraph, it can be seen that no action-state value is computed. Thus, the policy π cannot take any action based on the quality of each possible action given the current state of the environment. One of the most important algorithms proposed in the early years of the RL field is known as *Q-learning*. It updates the state-action value function, Q, as described in Equation 2.37, which looks pretty similar to the TD(0) state value update rule. For each step t of an episode, an action a_t is chosen according to the current state s_t following a given policy (*e.g.* ϵ-greedy policy, where the action with the highest state-action value is chosen with a probability ϵ). Then, r_{t+1} and s_{t+1} are observed and $Q_{t+1}(s_t, a_t)$ is computed. It has been proven that, given a few conditions on the sequence of the learning rate value are met and $t \to \infty$, the Q-function converges to $q*$ with probability 1 regardless of the policy being followed. This is due to the fact that the action taken one step ahead (*i.e.* for the state s_{t+1}) is chosen in order to maximize Q (*i.e.* $\max_a Q(s_{t+1}, a)$) (HABIB, 2019).

$$Q_{t+1}(s_t, a_t) = Q_t(a_t, s_t) + \alpha\left[r_{t+1} + \gamma \max_a Q_t(s_{t+1}, a) - Q_t(s_t, a_t)\right]. \tag{2.37}$$

In the Q-function update shown in Equation 2.37, the choice of the action a that maximizes $Q(s_{t+1}, a)$ is made using the same Q-function that is used to evaluate the chosen action. Thus, decisions are made based on an estimate function, which has an estimation error. In other words, if a wrong decision is made using the given Q-function, such a bad decision might be reinforced by the same Q-function. This issue causes the algorithm to exploit suboptimal decisions and, therefore, suboptimal policies, what slows down the convergence to the optimal q* (LAPAN, 2018).

In order to overcome this issue, the Double Q-Learning algorithm learns two different Q-functions (*i.e.* state-action value functions): Q_1 to guide the choice of actions, and Q_2 to evaluate the chosen actions, or vice-versa (HASSELT; GUEZ; SILVER, 2015). In this approach, for each step t of an episode, an action a_t is chosen using the policy $\epsilon - greedy$ with Q_1, Q_2, or the combination of both functions. Choosing one of the Q-functions at random or averaging their chosen actions in the case of a continuous action space are two possibilities for such a combination. After taking the action a_t, the agent observes r_{t+1} and s_{t+1}. Then, with a probability of 50%, Equation 2.38 is used to update the Q_1 function. Otherwise, Equation 2.39 is used to update Q_2.

$$Q_{1,t+1}(s_t, a_t) = Q_{1,t}(s_t, a_t) + \alpha \left[r_{t+1} + \gamma Q_{2,t}(s_{t+1}, \arg\max_a Q_{1,t}(s_{t+1}, a)) - Q_{1,t}(s_t, a_t) \right]. \tag{2.38}$$

$$Q_{2,t+1}(s_t, a_t) = Q_{2,t}(s_t, a_t) + \alpha \left[r_{t+1} + \gamma Q_{1,t}(s_{t+1}, \arg\max_a Q_{2,t}(s_{t+1}, a)) - Q_{2,t}(s_t, a_t) \right]. \tag{2.39}$$

In the Q-Learning-based methods presented above, it is necessary to try every sequence of combinations of states and values for the algorithm to be able to compute Q accurately for every possible case. In the real world this is impractical, since the set of possible states, and sometimes the set of actions, are usually huge. In order to give the Q-Learning capability of generalization and compute the Q-function for unseen combinations of states and actions, ML algorithms are applied to learn such a function from samples of experiences. In 2013, Mnih *et al.* published a groundbreaking paper where deep neural networks are used to approximate the Q-function in the Q-Learning algorithm (MNIH et al., 2013). In their approach, a neural network, called *Q-network*, receives the state of the environment encoded on an input vector of observed variables and outputs the state-action values for each possible discrete action (*i.e.* the output has one neuron for each possible action). The Q-network is trained through gradient descent with a batch of experiences, *i.e.* triplets of observed states, their taken actions, and the target Q values computed as $r_{t+1} + \gamma \max_a Q_t(s_{t+1}, a)$. The proposed algorithm was named *Deep Q-Netowrks* (DQN). In order to avoid the previously mentioned bias overestimation effect, in 2015, Mnih *et al.* proposed the *Double DQN* algorithm, a version of the DQN algorithm with two Q-networks. As in the Double Q-Learning algorithm, one of the networks is used to choose actions, while the other one evaluates the chosen actions (MNIH et al., 2015). Both algorithms presented superhuman performance in complex tasks such as playing several different Atari games without explicitly teaching the agent a single rule of the game.

Q-Learning-based algorithms are part of a wider group of RL algorithms called *model-free* methods. Different from *model-based* methods, in which actions rely on a model of the environment, model-free algorithms base their decisions on value functions that are learned from experience. Such a difference has been already discussed in this work when Temporal-Difference methods (*i.e.* model-free) were compared with Dynamic Programming (*i.e.* model-based). A model-free alternative for the Q-Learning algorithms is the family of Policy Gradient methods, which is presented in the next section.

2.3.2 Policy Gradient Methods

The RL algorithms presented so far are based on the idea of approximating value functions. The policy of the Q-Learning algorithm, for instance, is indirectly approximated by learning the state-action value function Q.

Policy gradient methods are techniques that directly learn a parameterized policy (LAPAN, 2018). In such methods, $\pi(a|s,\boldsymbol{\theta}) = Pr\{A_t = a|S_t = s, \boldsymbol{\theta_t} = \boldsymbol{\theta}\}$ is the probability of taking an action a in a state s when the policy's parameter vector is $\boldsymbol{\theta} \in \mathbb{R}^{d'}$, where d' is the number of policy's parameters. These algorithms learn $\boldsymbol{\theta}$ through gradient ascent on the gradient of some scalar performance measure $J(\boldsymbol{\theta})$ (see Equation 2.40), thus maximizing the agent's performance. It is important to mention that besides learning the policy's parameters, value functions can also be approximated in Policy Gradient methods. For example, $J(\boldsymbol{\theta_t})$ can be defined as $v_{\pi_{\boldsymbol{\theta}}}(s_0)$, which is the true value of the initial state given that the parameterized policy $\pi_{\boldsymbol{\theta}}$ is followed. However, as previously discussed, such a value must be learned throughout the search for the optimal policy. Therefore, approximation methods of state value functions such as the techniques we have been discussing so far can be used (SUTTON et al., 1999).

$$\boldsymbol{\theta_{t+1}} = \boldsymbol{\theta_t} + \alpha \nabla J(\boldsymbol{\theta_t}). \tag{2.40}$$

In order to ensure the exploration in the optimal policy search, the policy is usually required to be stochastic. It means that $\pi(a|s,\boldsymbol{\theta}) \in]0,1[$ for all a, s and $\boldsymbol{\theta}$. If the action space is discrete, the most common function used to define the probabilities of each action in the action space is the exponential softmax distribution over all possible actions. Such a function is shown in Equation 2.41, where $h(s,a,\boldsymbol{\theta}) \in \mathbb{R}$ is a parameterized numerical preference. These action preferences can be parameterized by a deep artificial neural network (ANN), where $\boldsymbol{\theta}$ is the weight vector of all connection between neurons. This parameterization is called *softmax in action preferences* (LAPAN, 2018).

$$\pi(a|s,\boldsymbol{\theta}) = \frac{e^{h(s,a,\boldsymbol{\theta})}}{\sum_b e^{h(s,b,\boldsymbol{\theta})}}. \tag{2.41}$$

Policy gradient methods also offer ways of computing actions in a continuous space. Instead of computing probabilities for each of the possible actions in a discrete action space, in a continuous space, the parameters of a continuous probability distribution must be learned. For example, a normal distribution can be used so that its mean and standard deviation must be learned. Like the softmax function in action preferences, an ANN can be used to approximate such a distribution (LAPAN, 2018). Equation 2.42 shows the probability of choosing an action a given a state s and the parameters $\boldsymbol{\theta}$, where $\pi \approx 3.14159$ (*i.e.* π is not the parameterized policy), and $\sigma(s, \boldsymbol{\theta})$ and $\mu(s, \boldsymbol{\theta})$ are the approximated standard deviation and the approximated mean of the distribution computed for the state s by a policy with parameters $\boldsymbol{\theta}$.

$$\pi(a|s, \boldsymbol{\theta}) = \frac{1}{\sigma(s, \boldsymbol{\theta})\sqrt{2\pi}} exp\left(-\frac{(a - \mu(s, \boldsymbol{\theta}))^2}{2\sigma(s, \boldsymbol{\theta})^2}\right). \tag{2.42}$$

In the following subsections, three closely related policy gradient methods with continuous action space that use ANNs to approximate their policies and value functions are presented. Due to the approximation of both policy and value functions, these algorithms can be classified as hybrid methods, since they inherit the policy approximation from policy gradient methods and the Q-function learning from the Q-Learning algorithm.

2.3.2.1 Deep Deterministic Policy Gradient

Proposed in 2015 by Lillicrap *et al.*, the Deep Deterministic Policy Gradient algorithm is a technique that uses ANNs to approximate policies for continuous action spaces and the Q-function (LILLICRAP et al., 2015). The Q-function is learned by minimizing the *mean-squared Bellman error* (MSBE) described in the Equation 2.43, where D is a set of experiences called *experience replay buffer*, Q_ϕ is the Q-function parameterized with the parameter vector $\boldsymbol{\phi}$, and $\pi_{\boldsymbol{\theta}}$ is the policy parameterized with the parameter vector $\boldsymbol{\theta}$. After calculating the MSBE for a given batch of experiences, the parameters $\boldsymbol{\phi}$ are adjusted through gradient descent. It is important to notice that the term between the outermost parenthesis is the same error expression used in the Q-function approximation equation of the Q-Learning algorithm, as presented in Equation 2.37. In DDPG, $Q_{\boldsymbol{\phi}}$ approximates the Q-function itself, while $\pi_{\boldsymbol{\theta}}$ learns to maximize Q *i.e.* it learns to take the best action given the current state and the current Q-function.

$$L(\boldsymbol{\phi}, D) = \frac{1}{|D|} \sum_{(s,a,r,s') \in D} \left(r + \gamma Q_{\boldsymbol{\phi}}(s', \pi_{\boldsymbol{\theta}}(s')) - Q_{\boldsymbol{\phi}}(s,a)\right)^2. \tag{2.43}$$

As previously mentioned, the performance $J(\boldsymbol{\theta})$ is used to adjust the policy parameters $\boldsymbol{\theta}$ with gradient ascent. In DDPG, such a performance is computed by averaging the state-action values of all actions taken for the states in the experience set D, as shown in Equation 2.44.

$$J(\boldsymbol{\theta}, D) = \frac{1}{|D|} \sum_{(s,a,r,s') \in D} Q_\phi(s, \pi_{\boldsymbol{\theta}}(s)). \tag{2.44}$$

In this method, a number of experiences is collected with a given policy $\pi_{\boldsymbol{\theta}}$ and added to the experience buffer replay. When it is time to update the policy and the Q-function, a set of experiences is randomly selected to perform the aforementioned updates. Then, Q-function is updated followed by the policy update. This process is repeated until a stopping criterium is met. When the update ends, the current iteration ends and the agent needs to gather new experiences with the new updated policy starting a new iteration. The whole training process ends when another stopping criterium is met, usually when the learning process converges.

In order to avoid bias overestimation, a second network can be used for each one of the ANNs used in DDPG, one for $Q_{\boldsymbol{phi}}$ and another one for $\pi_{\boldsymbol{\theta}}$. These networks are called *target networks* and are used in the target calculation of the MSBE computation as shown in Equation 2.45. In this formula, the MSBE equation is rewritten with the target networks $Q_{\boldsymbol{\phi}_{target}}$ and $\pi_{\boldsymbol{\theta}_{target}}$. These networks hold an approximate copy of the parameters of their respective main networks, but they are updated from time to time after the main network policy is updated. Such an update is made in a frequency that must be previously defined. Such a mechanism creates a delay between the main and the target networks.

$$L(\boldsymbol{\phi}, D) = \frac{1}{|D|} \sum_{(s,a,r,s') \in D} \Big(r + \gamma Q_{\boldsymbol{\phi}_{target}}(s', \pi_{\boldsymbol{\theta}_{target}}(s')) - Q_{\phi}(s,a) \Big)^2 . \tag{2.45}$$

It is important to mention that instead of just copying the parameters from the main to the target networks, the target networks are updated by polyak averaging as shown in Equations 2.46 and 2.47, where ρ is a hyperparameter that defines how much from the previous target policies are kept when they are updated.

$$\boldsymbol{\phi}'_{target} = \rho \boldsymbol{\phi}_{target} + (1-\rho)\boldsymbol{\phi}, \tag{2.46}$$

$$\boldsymbol{\theta}'_{target} = \rho \boldsymbol{\theta}_{target} + (1-\rho)\boldsymbol{\theta}. \tag{2.47}$$

Finally, in order to generate diversity in the policy training process, a mean-zero Gaussian noise $\epsilon \sim N$ with a user-defined standard deviation is added to the taken actions. This noise works as a regularizer in the learning process, avoiding the policy to get stuck in local optimum. Besides, the actions are clipped within the interval $[a_{low}, a_{high}]$ to keep the taken actions within a valid interval. However, this is done only during the training process, which means that when a trained policy is tested, the raw output of the policy network must be used instead.

2.3.2.2 DDPG with Distributed Prioritized Experience Replay

The experience replay is the mechanism of choosing experiences in the replay buffer to learn from. For instance, in the original DDPG algorithm, this selection is made randomly. In 2015, Schaul *et al.* proposed the *prioritized experience replay* (SCHAUL et al., 2015). Based on the concepts of prioritized sweeping proposed by More and Atkeson in 1993 (MOORE; ATKESON, 1993), in the prioritized experience replay, the experiences from which the agent must learn are chosen with different probabilities. These probabilities are calculated according to how much information the agent may gain from each experience. It means that the experiences on which the neural network-based approximators show the highest losses are more likely to be chosen. The objective of such a mechanism is to speed up the learning process of RL algorithms and make these algorithms more sample-efficient (*i.e.* require less training experiences to achieve a certain level of performance), which is one of the biggest challenges in the RL field (KARIMPANAL, 2020). Since its proposal, prioritized experience replay has been successfully used in several well established algorithms, such as DQNs (WANG et al., 2015). In 2017, Hessel *et al.* showed that such a prioritization is the most important component for the success of many state-of-the-art RL algorithms (HESSEL et al., 2017).

Neural networks have become increasingly costly to train. Distributed stochastic gradient descent has been used in many deep learning frameworks to speed up the training process (HORGAN et al., 2018). In these implementations, the parameter updates can be done synchronously (KRIZHEVSKY, 2014) or asynchronously (DEAN et al., 2012). In the synchronous mode, all gradients are calculated by multiple workers and applied at once. On the other hand, in the asynchronous mode, once any worker finishes calculating its portion of gradients, it updates the weights of the network without waiting for the others.

Distributed implementation have also been proposed for Reinforcement Learning. In most of these implementations, many workers run in parallel and fill the experience buffer much faster than the single-worker versions of the algorithm (LIANG et al., 2017). A few authors have also proposed a distributed update of the policy and value function parameters.

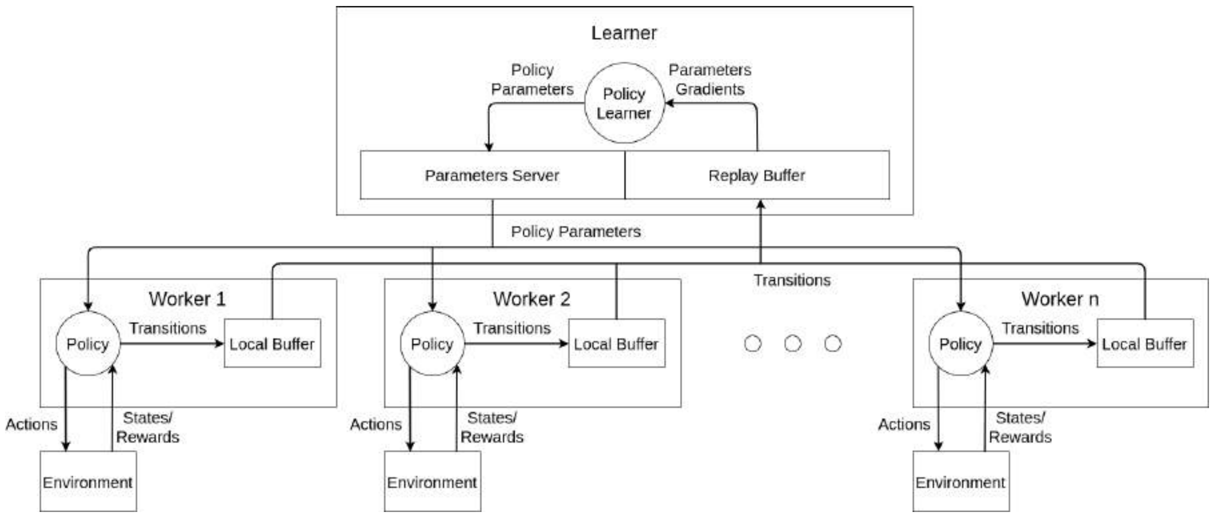
In 2018, Horgan *et al.* proposed a distributed prioritized experience replay mechanism for RL algorithms, what they called Ape-X (HORGAN et al., 2018). In this approach, the experiences are collected by multiple independent workers running in parallel. Thus, experiences can be collected more efficiently than the algorithms that use a single agent.

Each agent stores the collected experiences in a local buffer and, when it is full, such experiences and their precomputed corresponding priorities are transferred to a centralized shared memory. It is important to highlight that these workers run without any point of synchronization. Therefore, the centralized shared memory is partitioned into shards, one for each worker to store its collected experiences.

A central thread that runs the training algorithm itself periodically and asynchronously samples a few prioritized experiences in order to calculate their gradients and update its parameters. Such a thread is called *learner.* These parameters are stored in a parameter server. After sending the experiences to the centralized learner, each worker sends a request of policy update to this server. When the request is attended, the worker flushes its local buffer and starts collecting new experiences. When the centralized replay buffer is full, some experiences are chosen to be removed. It can be done by just removing the oldest ones or by selecting random experiences, for instance.

The generality of Ape-X was assessed by implementing and testing distributed versions of DQN (*i.e.* Ape-X DQN), and DDPG (*i.e.* Ape-X DDPG). Figure 7 shows the Ape-X architecture.

Figure 7 – The Ape-X architecture.



Source: Produced by the author.

2.3.2.3 Twin Delayed DDPG

As already mentioned, the target networks used in DDPG are intended to reduce the bias overestimation, where bad actions are exploited. However, such an issue was not completely removed from the algorithm. In order to improve even more the performance of DDPG, in 2018 Fujimoto *et al.* proposed a few changes in the algorithm. This new method was named Twin Delayed DDPG (TD3) algorithm (FUJIMOTO; HOOF; MEGER, 2018).

Three changes were made in the original DDPG algorithm. The first one was the addition of a clipped noise in the action taken by the target policy $\pi_{\boldsymbol{\theta}_{target}}$ during the MSBE computation. The noise is clipped within the interval $[-c, c]$, where c is a constant that must be previously defined. Such a mechanism is intended to smooth the target policy, since wrong actions with peak values might make the algorithm to quickly exploit them and get stuck in a local optimum. Therefore, the clipping mechanism works as a regularizer of the added noise when it causes such an undesired effect. In addition, the final taken action value is clipped within the interval $[a_{low}, a_{high}]$, just like DDPG and for the same reason.

The second change is the use of two neural networks to compute the Q-function and the use of two target networks. Thus, instead of calculating the Q value with a single network, two networks compute separately and the minimum Q value is chosen. This choice is made in order to avoid very high Q values for bad decisions, what reduces even more an eventual bias overestimation. Equations 2.48 and 2.49 show the loss calculation for networks 1 and 2, respectively.

Unlike in the Double Q-Learning algorithm, both networks are updated in every step of the episode. Analogously to DDPG, the policy is updated by gradient ascent. However, only $Q_{\boldsymbol{\phi}_1}$ is used to evaluate the taken actions.

$$L(\boldsymbol{\phi}_1, D) = \frac{1}{|D|} \sum_{(s,a,r,s') \in D} \left(r + \gamma \min_{i=1,2} Q_{\boldsymbol{\phi}_{i,target}}(s', \pi_{\boldsymbol{\theta}_{i,target}}(s')) - Q_{\boldsymbol{\phi}_1}(s,a) \right)^2 . \tag{2.48}$$

$$L(\boldsymbol{\phi}_2, D) = \frac{1}{|D|} \sum_{(s,a,r,s') \in D} \left(r + \gamma \min_{i=1,2} Q_{\boldsymbol{\phi}_{i,target}}(s', \pi_{\boldsymbol{\theta}_{i,target}}(s')) - Q_{\boldsymbol{\phi}_2}(s,a) \right)^2 . \tag{2.49}$$

Finally, the third and last change in the original DDPG algorithm is the addition of a delay between the updates of the policy and the Q-function, where the Q-function will be updated more often. For example, for each update of the policy network, the Q-function can be updated twice, what is recommended by the authors of TD3.

Like the Ape-X DDPG algorithm, the distributed implementation of TD3 can be implemented using the architecture shown in Figure 7. Such an implementation is available in (LIANG et al., 2017). However, unlike the Ape-X DDPG, TD3 uses synchronous workers and chooses random transitions during the experience replay. It means that the centralized replay buffer is updated only when all workers have finished collecting experiences.

2.4 POPULATION BASED TRAINING

Machine Learning algorithms usually require the adjustment of a large set of hyperparameters. The automatic adjustment of hyperparameters is essential to ease the use of such techniques. Proposed in 2017 by Jaderberg *et al.*, Population-Based Training (PBT) is an asynchronous population-based optimization algorithm that was designed to efficiently control the hyperparameters of ML algorithms during the training process (JADERBERG et al., 2017; LI et al., 2019). The idea of the algorithm is to run multiple training processes in parallel, each one with different values for the hyperparameters. Each training process runs inside an asynchronous worker. From time to time, the hyperparameters and parameters are exchanged between workers and the hyperparameters are perturbed, thus changing them throughout the training process. The online adjustment of hyperparameters is especially useful when the learning problem is highly non-stationary, which is the case for RL algorithms.

Algorithm 6 shows the pseudocode for the PBT algorithm. It is important to note that this is a general description of the PBT algorithm and the exploitation and exploration operators must be defined for each application, as well as the stopping conditions.

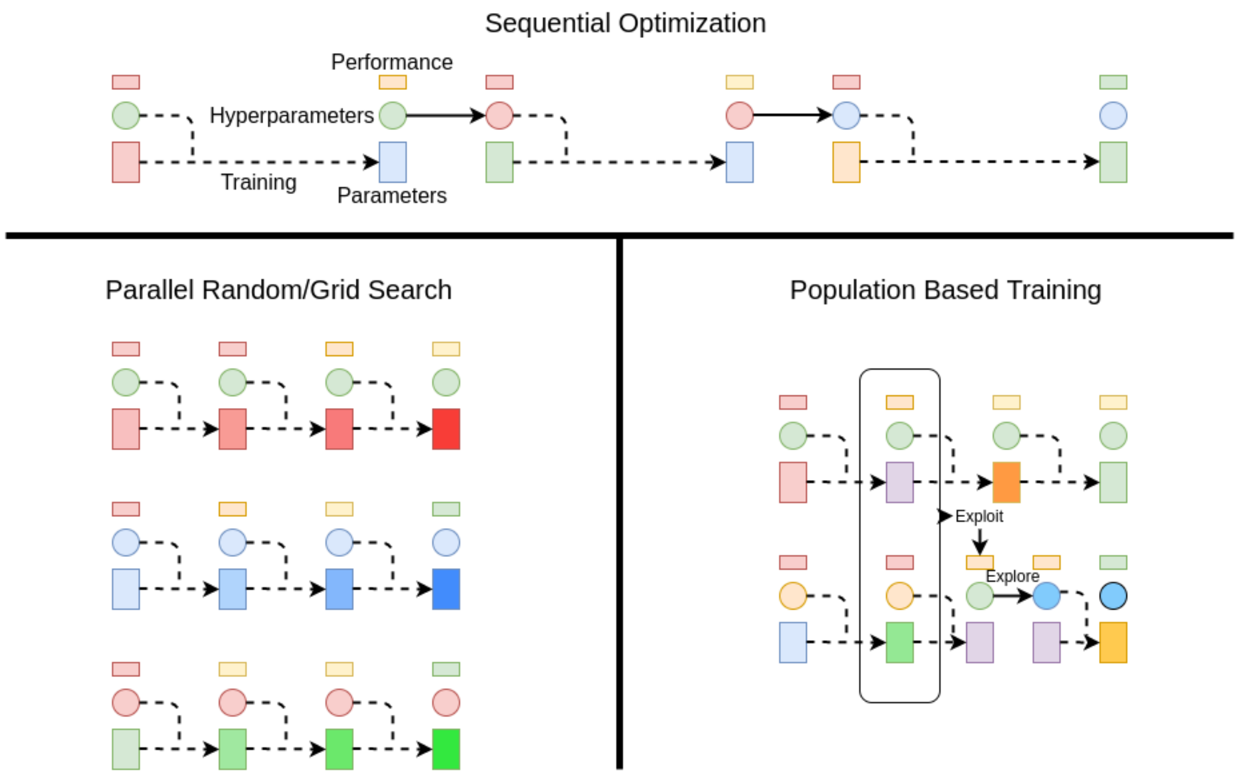
Algorithm 6: Pseudocode for Population Based Training.

```
Input: Initialize the population of workers and their respective parameters and
       hyperparameters.
Output: Best pair of parameters and hyperparameters vectors ever found.
1 foreach worker running in parallel and asynchronously do
2     while stopping condition is not met do
3         RUN its internal training process starting with the current
            hyperparameters
4         if a given condition is met then
5             EXCHANGE parameters and hyperparameters with another worker
                (exploitation operator)
6             PERTURB the hyperparameter vector (exploration operator)
7 return Best pair of parameters and hyperparameters vectors ever found.
```

Figure 8 illustrates the PBT algorithm and compares it to the Sequential Optimization and the Parallel Random/Grid Search method. In this figure, each rectangle represents a set of values for the parameters of a given model, while each circle represents a set of values for its hyperparameters. Similar colors between rectangles means similar parameters, and consequently similar colors between circles means similar hyperparameters. Each pair of rectangles and circles represents a state of a training model. The horizontal bars above the circles represent the quality of the models, which means that the bars in red are the worst ones, while the bars in green are the best ones. Each horizontal sequence of models connected by dashed lines represents a training process. It means that in the Sequential Optimization (a), the only training process is executed sequentially, while in the Parallel Random/Grid Search (b) and Population Based Training (c), multiple training processes are performed in parallel. The difference between the Parallel Random/Grid Search and the Population Based Training is the communication between training processes, represented by the vertical rectangle in Figure 2.c. In this figure, the parameters and hyperparameters from the upper training process are copied into the model in the lower training process (exploitation). Then, the hyperparameters of the copied model are perturbed (exploration), what is represented by the transition from the green circle to the blue circle. Notice that the color of the rectangle in purple has not changed during the exploration phase, since only the hyperparameters are perturbed (JADERBERG et al., 2017; LI et al., 2019).

It can be seen that PBT inherits the best characteristics of both techniques: perform multiple parallel searches like parallel random/grid search, and change the hyperparameters on-the-fly like sequential optimization. Running a population of PBT workers has the obvious advantage of avoiding getting stuck in local optima, while changing hyperparameters on-the-fly has the advantages of any control algorithm. Together with the communication between PBT workers, these features allow the PBT algorithm to be very efficient and effective, since it strongly benefits from distributed platforms and concentrates computing resources on promising regions of both the hyperparameter and parameter search spaces. The PBT's original paper showed that RL algorithms can strongly benefit from the technique (JADERBERG et al., 2017; LI et al., 2019).

Figure 8 – Graphical representation of the PBT algorithm and comparisons with parallel random/grid search, and sequential methods.



Source: Produced by the author based on (JADERBERG et al., 2017).

3 LITERATURE REVIEW

"Smart people learn from everything and everyone, average people from their experiences, stupid people already have all the answers."

Authorship supposedly assigned to Socrates.

The parameter control problem for metaheuristics has received an increasing attention. For example, since the propsosal of the Particle Swarm Optimization algorithm (PSO) (KENNEDY; EBERHART, 1995), the adaptation of its parameters has been widely explored (*e.g.* (SHI; EBERHART, 1998)(DONG et al., 2008)(CHEN; LI; LIAO, 2009)(NICKABADI; EBADZADEH; SAFABAKHSH, 2011)(XU, 2013)). Besides the PSO algorithm, Genetic Algorithms (GA) and Differential Evolution (DE) have been also heavily explored in the literature (PARPINELLI; PLICHOSKI; SILVA, 2019). However, most of these studies are experimental, even though theoretical analyses have also been made for the most common algorithms (*e.g.* PSO, GA, DE, ES, etc). A study about adaptive versions of the PSO algorithm was published in 2016 by Harrison *et al.* (Harrison; Engelbrecht; Ombuki-Berman, 2016). In this paper, the authors analyzed the convergence behavior of eight variants of the PSO algorithm. Dang *et al.* (DANG; LEHRE, 2016) and Doerr *et al.* (DOERR; WITT; YANG, 2018) performed rigorous runtime analyses of evolutionary algorithms with self-adaptive mutation rates and proved their effectiveness. In 2018, Qian *et al.* showed that adaptive sample sizes for evolutionary algorithms solving noisy problems work better than sampling with fixed size (QIAN et al., 2018). In 2019, Del Ser *et al.* published a comprehensive survey on the most recent advances in evolutionary and swarm-based algorithms, where the most relevant theoretical analysis of these algorithms can be found (SER et al., 2019).

A few literature reviews on parameter control and/or tuning methods have been published since the 1990's. In 1999, Eiben *et al.* organized the field of parameter adjustment for metaheuristics, dividing the solutions for the first time into tuning and control methods (EIBEN; HINTERDING; MICHALEWICZ, 1999). Also, the authors of this study classified the control methods among deterministic, adaptive and self-adaptive algorithms. Besides, the discussed techniques were grouped according to the adapted parameter (*i.e.* representation of candidate solutions, mutation rate, crossover rate, etc).

In 2012, Zhang *et al.* published a survey that focus on adaptive mechanisms for evolutionary algorithms with a more detailed classification scheme (ZHANG et al., 2012). They classified the reviewed techniques according to three different taxonomies: adaptation objects (*i.e.* What is adapted?), which are subdivided into control parameters, evolutionary operators, population structure, and "others"; adaptation evidences (*i.e.* What are the evidences that guide the adaptation mechanism?), which are subdivided into deterministic factors such as time, fitness values, population distribution, and fitness values with population distribution; and adaptation methods (*i.e.* How are these objects adapted?), which are subdivided into simple rules-based mechanisms, coevolution, entropy-based control, and fuzzy control. Also, in order to better illustrate the analyzed problem, the authors have detailed a state-of-the-art adaptive Genetic Algorithm (Zhang; Chung; Lo, 2007) and an adaptive version of the Particle Swarm Optimization algorithm (Zhan et al., 2009).

In 2013, Jordehi and Jasni presented a survey about parameter adaptation for PSO (JORDEHI; JASNI, 2013). As in the previously cited survey, the authors divided the algorithms according to the adapted parameters. Also, the paper presented and discussed parameter-free versions of the PSO algorithm.

In 2015, Karafotias *et al.* (KARAFOTIAS; HOOGENDOORN; EIBEN, 2015b) updated the survey published by Eiben *et al.* in 1999 (EIBEN; HINTERDING; MICHALEWICZ, 1999). Also, the authors presented a new way to classify the existing parameter adjustment mechanisms into four groups: control methods tailored to application, out-of-the-box control method, tuning methods defining static values, and static values defined by intuition or convention (*i.e.* no automatic parameter adjustment mechanism used). In order to group the retrieved papers, the authors classified them according to the adapted parameters. Besides, this was the first survey that classified a group of techniques as out-of-the-box algorithms. However, it is important to highlight that some of the discussed techniques could not be considered as actual out-of-the-box methods, since they cannot be truly applied to every possible parameter of metaheuristics. In this study, the concept of out-of-the-box methods seems to have been a little bit "relaxed".

In 2016, Moser and Aleti published a systematic literature review about automatic parameter adjustment (ALETI; MOSER, 2016). To the best of our knowledge, this is the only systematic literature review about parameter adjustment that has been published so far. The authors did an outstanding job grouping a considerable amount of papers according to different aspects. However, the authors did not consider classifying the studies according to whether the proposed method is out-of-the-box or not. Thus, the aspects that need to be analyzed when talking about out-of-the-box approaches are missing in this study.

In 2017, Guan *et al.* published a survey focused on population size adaptation for Evolutionary Algorithms (GUAN; YANG; SHENG, 2017). In the first part of the paper, the authors have divided the articles into theoretical and experimental studies. Besides, the experimental studies were further divided into deterministic methods, adaptive methods and self-adaptive methods. In the second part of the study, the authors compared the performances of the algorithms that have been considered as part of the state-of-the-art of the field. To the best of our knowledge, this has been the only study that presented experiments with comparisons between the most relevant techniques discussed in the review itself.

In 2019, Parpinelli *et al.* presented a survey about parameter control for evolutionary and swarm-based algorithms. They presented an algorithm classification scheme adapted from Eiben *et al.* (EIBEN; HINTERDING; MICHALEWICZ, 1999) and Zhang *et al.* (ZHANG et al., 2012). The techniques were classified into online and offline methods. The online methods were further classified into deterministic, adaptive and aggregated algorithms. Finally, the adaptive techniques were classified even further in simple rules, fuzzy control, learning automata, entropy-based method and others. This is the study that best covers the parameter control approaches for swarm-based algorithms.

In 2020, Huang *et al.* published a survey focused on parameter tuning methods (Huang; Li; Yao, 2020). In this study, the authors divided the methods into 3 types: Simple Generative-Evaluate Methods, High-Level Generate-Evaluate Methods, and Iterative Generate - Evaluate Methods. The later was further divided into 4 groups: Experimental Design based Methods, Numerical Optimization based Methods, Heuristic Search based Methods, and Model-based Optimization Methods. Even though this review was not systematically made, it presents a comprehensive analysis of the cited techniques, where the most relevant tuning methods proposed up to its date of publication are highlighted (*e.g. (HUTTER et al., 2009)(BALAPRAKASH; BIRATTARI; STÜTZLE, 2007b)(NANNEN; EIBEN, 2007)*).

Even though these surveys are quite relevant to the problem at hand and bring useful insights on future directions, none of them is focused on out-of-the-box control mechanisms. Besides, the only study that mentioned the idea of out-of-the-box methods and classified some techniques accordingly was the one published by Karafotias *et al.* in 2015 (KARAFOTIAS; HOOGENDOORN; EIBEN, 2015a), even though the inclusion of some studies in this group are indeed questionable. Also, it is important to note the lack of systematic literature reviews in the field. Therefore, in 2021 we have published the first systematic literature review on out-of-the-box parameter control methods for evolutionary and swarm-based algorithms (LACERDA et al., 2021). The main objective of this literature review was to identify and discuss out-of-the-box approaches for parameter control for EA and swarm-based algorithms, and present the trends and the main challenges of such a field. Also, a detailed explanation of each method and comparisons between them were presented. It is important to highlight that none of the previous surveys presented such a detailed analysis of out-of-the-box methods. In the next sections, the aforementioned systematic literature review is presented.

3.1 METHODOLOGY

This section presents the research questions answered in this systematic literature review and its methodology.

3.1.1 Research Questions

The *research questions* are listed below.

- RQ1: Which methods for out-of-the-box parameter control have been used in the literature?
- RQ2: Which methods for out-of-the-box parameter control are self-adaptive on their own parameters?
- RQ3: Which methods adapt their search to the limitations of the underlying hardware, time budget, or maximum number of fitness evaluations?
- RQ4: What are the main challenges and future trends for out-of-the-box parameter control for EA and SI?

It is important to mention that RQ2 was posed due to the importance of diminishing the need for manual parameter adjustment. Also, RQ3 is an important question since these methods are usually compute-intensive.

3.1.2 Types of accepted studies

This literature review is concerned with any study published in a scientific journal or a conference proceeding that matches with any of the following *inclusion criteria*:

1. Articles that develop or improve mechanisms for out-of-the-box parameter control for EA and SI;
2. Articles that apply any general mechanism of parameter control in EA and SI to solve practical problems;

This paper is **not** concerned with studies that match with any of the following *exclusion criteria*:

1. Articles that are not written in English;
2. Articles that are not published in any scientific journal nor conference proceeding;
3. Articles where the proposed method is tailored to an application;
4. Articles where operator selection methods are proposed;
5. Articles where the proposed controller is suitable for anything but EA or SI.

3.1.3 Study identification and selection

In order to identify and select articles for this literature review, a search string was defined and used in the following databases: ACM Digital Library, Scopus (includes other traditional databases, such as Science Direct, Springer Link and IEEE Xplore), Web of Science and arXiv.org. Articles were searched with the following search string in their title and abstract: *("parameter control" OR "parameter adaptation" OR "self-adaptive") AND ("evolutionary computation" OR "swarm intelligence" OR "metaheuristic" OR "evolutionary algorithm").*

The identification and selection of studies was the first step performed in the review process. This stage was divided into two phases. In the first phase, the aforementioned search string was used in the databases to retrieve all articles published until January 20th, 2020. Then, all duplicates were removed. In the second phase, the inclusion and exclusion criteria were applied to the remaining studies by analyzing only titles and abstracts. In case the number of papers retrieved from the databases is low, the inclusion and exclusion criteria are also applied to the title and the abstract of the references of the selected papers in the current phase.

3.1.4 Data Extraction

In the data extraction process, the relevant information from the selected papers in the previous phase were extracted. To extract such an information, these articles had to be fully analyzed. In order to avoid any bias, two researchers worked together in this phase, discussing the divergences until it came to a common agreement. The researchers focused on the identification of the methods used to build the controllers, the controlled optimization algorithms, the controlled parameters, and the optimization problems used in the experiments.

3.2 RESULTS AND DISCUSSIONS

This section presents the results of our systematic literature review, which is divided into two parts: Section 3.2.1, that presents an overview of the collected studies, and Section 3.2.2, where these studies are detailed.

3.2.1 Results Overview

After using the search string in the databases mentioned in the Subsection 3.1.3, a total of 4449 papers were retrieved. Table 1 shows the distribution of retrieved papers for each database. After removing duplicates, 3983 articles remained. The application of the inclusion and exclusion criteria removed another 3933 articles. Thus, 50 studies remained in the list of selected papers. Then, these articles were further analyzed in the extraction phase.

Table 1 – Number of papers retrieved with each database.

Database	Retrieved documents
ACM Digital Library	1458
arXiv.org	38
Scopus	2172
Web of Science	781

Source: Produced by the author.

Given that in the selection phase only the title and the abstract have been analyzed, in a few cases it was not clear whether the inclusion or exclusion criteria match or not. For these cases, we have decided to pass them to the data extraction phase for a deeper analysis. Thus, after deeply analyzing each selected paper, another 37 studies were excluded. In the end, 0.4% of the selected papers matched the criterion 1, 3%, matched the criterion 2, 59.2% matched the criterion 3, 5.2% matched the criterion 4, and 37.7% matched the criterion 5. The sum of the percentages for the aforementioned criteria is not 100%, since each paper can match to multiple criterion.

Finally, an amount of 13 articles have been selected to compose this literature review. We considered such a number of accepted studies very low. Thus, the inclusion and exclusion criteria were also applied to their references, adding 2 other papers to the list of accepted studies, totaling 15 selected articles.

Table 2 shows an overview of the results of the extraction phase. As mentioned in the Section 3.1.4, the researchers involved in this phase focused on the identification of the methods used to build the controllers, the controlled algorithms used in the experiments, the controlled parameters, and the optimization problems. Due to space limitations in the Table 2, some of the names of the optimization algorithms and problems were written in acronyms (some of them have been already introduced in this work): Genetic Algorithms (GA), Evolution Strategies (ES) (BEYER, 1995), Differential Evolution (DE), Particle Swarm Optimization, Tabu Search (TS) (GLOVER; LAGUNA, 1997), Quadratic Assignment Problem (QAP), Royal Road Problem (RRP), Component Deployment Problem (CDP), Optimal Ordering of Tables (OOT), and Traveling Salesman Problem (TSP).

It can be observed that the majority of the studies have applied RL or other predictive methods as parameter control algorithms. Both methods were used in 66,67% of the analyzed studies. Besides, RL was the most common paradigm used in the last 6 years of publications (*i.e.* 4 out of 5 studies). This is probably due to the recent huge success of RL algorithms in highly complex scenarios that were once considered intractable, such as beating a world-class player of Go (SILVER et al., 2016).

The predictive approaches use algorithms for time-series analysis to approximate the chance of success of each possible value for each parameter. The four papers that applied such an approach were proposed by the same research group and are actually a sequence of improvements of an initial algorithm published between 2011 and 2014.

Table 2 – Overview of the data extraction process.

Article	Year	Control Method	Controlled Algorithms	Controlled Parameters	Optimization Problems
(ROST; PETROVA; BUZDALOVA, 2016)	2016	Reinforcement Learning	GA	Mutation step-size	Continuous benchmark functions.
(KARAFOTIAS; HOOGENDOORN; EIBEN, 2015a)	2015	Reinforcement Learning	ES and GA	All parameters	Continuous benchmark functions.
(ALETI et al., 2014)	2014	Predictive Parameter Control	GA	All parameters	QAP and RRP.
(KARAFOTIAS; HOOGENDOORN; WEEL, 2014)	2014	Reinforcement Learning	ES and GA	All parameters	Continuous benchmark functions.
(KARAFOTIAS; EIBEN; HOOGENDOORN, 2014)	2014	Reinforcement Learning	ES and GA	All parameters	Continuous benchmark functions.
(ALETI; MOSER, 2013)	2013	Predictive Parameter Control	GA	Crossover and Mutation rates	QAP and CDP.
(BIELZA; POZO; LARRAñAGA, 2013)	2013	Bayesian Networks	GA	Crossover and Mutation rates	OOT.
(ALETI; MOSER; MOSTAGHIM, 2012)	2012	Predictive Parameter Control	GA	Crossover and Mutation rates	QAP, RRP and CDP.
(KARAFOTIAS; SMIT; EIBEN, 2012)	2012	Reinforcement Learning	ES	Mutation step-size	Continuous benchmark functions.
(LEUNG; YUEN; CHOW, 2012)	2012	Surrogate Models	GA, DE and PSO	Crossover rate, differential amplification factor, C1, C2, and inertia weight.	Continuous benchmark functions.
(CHATZINIKOLAOU, 2011)	2011	Agent-based Genetic Algorithm	GA	Mutation rate	Rastrigin function.
(ALETI; MOSER, 2011)	2011	Predictive Parameter Control	GA	Crossover and mutation rates	RRP and CDP.
(MATURANA; SAUBION, 2008)	2007	Fuzzy Logic	GA	Mutation and crossover rates.	QAP.
(EIBEN et al., 2007)	2006	Reinforcement Learning	GA	All parameters	Benchmark functions.
(AINE; KUMAR; CHAKRABARTI, 2006)	2006	Dynamic Programming	GA	Crossover and mutation rates	TSP.

Source: Produced by the author.

The RL-based approaches are all based on Temporal Difference, with exception of (KARAFOTIAS; SMIT; EIBEN, 2012), which was the very first study that formalized the use of such a paradigm for parameter control. These studies were strongly influenced by (KARAFOTIAS; EIBEN; HOOGENDOORN, 2014) and were mostly written by the same research group as well.

Regarding the controlled algorithms, it is important to notice that, with the exception of (LEUNG; YUEN; CHOW, 2012), all studies used only EAs as controlled algorithm. Therefore, there is a lack of studies that evaluated out-of-the-box control methods in swarm-based algorithms.

Even though these methods are supposed to work with any algorithm and parameter, in the majority of the papers only part of the parameters of the chosen metaheuristics is controlled. Also, it can be noted that the crossover and the mutation rates/step-size are the ones that are mostly chosen to be controlled. The remaining parameters are kept constant and must be previously defined by the user. In these studies, the authors defined these constant parameters through tuning methods or in an ad-hoc manner. Controlling part of the parameters set makes the problem easier to deal with, since the parameter space becomes simpler.

It is important to highlight that just a small part of the analyzed studies control the population size. The high complexity of the population size control was verified by Aleti *et al.* in 2014 (ALETI et al., 2014). Besides that, they used only EAs as controlled algorithms. This is probably due to the fact that controlling the population size of a swarm-based algorithm requires the creation of an addition/removal mechanism of candidate solutions. On the other hand, the concepts of killing or giving birth to candidate solutions are inherent to any EA.

Regarding the optimization problems, no real-world instances have been used in any of the studies. In 8 of the 15 selected papers, the experimental benchmark sets were composed of well-known benchmark functions. In the other studies, the control methods were tested on benchmark instances of traditional combinatorial or integer-mixed optimization problems.

Table 3 describes the results of the quality assessment of the accepted articles. For this purpose, a questionnaire was created with the following questions, to which three answers are possible: Yes (Y), Partially (P) and No (N). This questionnaire had to be created for this literature review because there was no available questionnaire with such a purpose. Questions 1, 2 and 3 try to assess the quality of the description of the proposed method in the paper. Also, they are intended to check if the authors are clear about the downsides of their proposed controllers. Questions 4, 5, 6, 7 and 8 are intended to assess the quality of the experimental methodology. It is important to mention that the values defined for the question 5 are based on the number of functions available in the benchmark set of the CEC 2017 Competition on Constrained Real-Parameter Optimization (AWAD et al.,

Table 3 – Quality assessment of the accepted studies.

Article	Year	Q1	Q2	Q3	Q4	Q5	Q6	Q7	Q8	**Total**
(KARAFOTIAS; EIBEN; HOOGENDOORN, 2014)	2014	1	1	0.5	1	0.5	1	1	1	**7**
(KARAFOTIAS; HOOGENDOORN; EIBEN, 2015a)	2015	0.5	1	0.5	1	0.5	1	1	1	**6.5**
(KARAFOTIAS; HOOGENDOORN; WEEL, 2014)	2014	0	1	1	1	0.5	1	1	1	**6.5**
(ALETI et al., 2014)	2014	0	1	1	1	0	1	1	1	**6**
(ALETI; MOSER; MOSTAGHIM, 2012)	2012	0	1	1	1	0	0.5	1	1	**5.5**
(LEUNG; YUEN; CHOW, 2012)	2012	1	1	0.5	1	1	0	1	0	**5.5**
(ALETI; MOSER, 2011)	2011	1	0	1	1	0	0.5	1	1	**5.5**
(ALETI; MOSER, 2013)	2013	0	0	1	1	0	0.5	1	1	**4.5**
(MATURANA; SAUBION, 2008)	2007	1	1	1	0	1	0.5	0	0	**4.5**
(BIELZA; POZO; LARRAñAGA, 2013)	2013	1	0	1	0	0	0.5	0	1	**3.5**
(ROST; PETROVA; BUZDALOVA, 2016)	2016	0	0	1	1	0	0	0	1	**3**
(KARAFOTIAS; SMIT; EIBEN, 2012)	2012	1	0	1	0	0	0	0	1	**3**
(CHATZINIKOLAOU, 2011)	2011	0	1	1	0	0	0	0	1	**3**
(EIBEN et al., 2007)	2007	1	0	1	0	0	1	0	0	**3**
(AINE; KUMAR; CHAKRABARTI, 2006)	2006	1	0	1	0	1	0	0	0	**3**
Total		**8.5**	**8**	**13.5**	**9**	**4.5**	**7.5**	**8**	**11**	**70**

Source: Produced by the author.

2016), which are 29.

1. Have the authors described the weaknesses of the proposed technique? (Y/P/N)
2. Have the authors described the threats of the application context? (Y/P/N)
3. Have the authors described the parameter control algorithm? (Y/P/N)
4. Have the authors applied hypothesis tests to draw robust conclusions from the results? (Y/N)
5. Have the authors tested the proposal on a big set of benchmark problems (N: 1-10 instances; P: 11-20 instances; Y: 21 instances or more)?
6. Did the proposed approach controls all parameters of the controlled algorithm on the experiments? (N: controls 1 parameter; P: controls more than 1, but not all of them; Y: controls all parameters)
7. Have the authors tested the proposal on multiple optimization algorithms? (Y/N)
8. Have the authors compared the proposed method with multiple control methods? (Y/N)

As it can be observed, the studies with the highest standards are the most recent ones, except for Rost *et al.* (ROST; PETROVA; BUZDALOVA, 2016). It shows that this field is becoming more and more mature, as any other relatively new scientific field. However, it can also be observed that, among the 7 highest scoring papers, 6 of them were written by the same two research groups.

Question 5 was the least scored, which means that there is a lack of studies that applies their approaches to a large benchmark problem set. The second least scored question is question 6, which meaning that there is also a lack of studies where all parameters are controlled by the algorithm, especially population size, as mentioned before. Questions 1 and 2 are not well scored as well, since it is not common to find, in any field, papers that openly talk about their weaknesses and threats.

The highest scored questions are 3, 8 and 4. Question 3 talks about how well the control method is explained in the study. For this question, the score was already expected to be high. Question 8 tells whether the proposed control method was compared to other control methods or not. Question 4 quantifies how often hypothesis tests have been used. Even though this is a mandatory tool to draw correct conclusions from experiments with non-deterministic algorithms, this has never been a common practice in the field of EA and swarm-based algorithms. Thus, the high score observed in such a question is surprising.

Overall, all studies together scored 70 points. This is approximately 58.4% of the total of 120 points. Even though it means that there is a lot to improve on the quality of the papers in this field, the latest papers (except Rost *et al.*, 2016 (ROST; PETROVA; BUZDALOVA, 2016)) presented very high standards, especially when compared to the oldest ones. It means that this field has shown studies with an increasing good quality. Thus, it is expected that future studies will present high standards as well. It is important to mention that, despite some of the selected papers presented low quality standards, they still present relevant contributions to the field.

In the next section, the aforementioned accepted studies are more detailed.

3.2.2 Detailing accepted studies

The studies detailed in this section are classified according to the approach used to implement the parameter controller. Since there are methods that were used only once, these studies are grouped together as "Others". In order to ease the visualization of the whole picture, Table 4 groups the studies by the approach used to develop the controller and the controlled parameters. Due to space limitations on the Table 4, the references of the papers were replaces by numbers as depicted below:

1. (AINE; KUMAR; CHAKRABARTI, 2006);

2. (ALETI et al., 2014);

3. (ALETI; MOSER, 2011);

4. (BIELZA; POZO; LARRAñAGA, 2013);

5. (ALETI; MOSER, 2013);

6. (ALETI; MOSER; MOSTAGHIM, 2012);

7. (CHATZINIKOLAOU, 2011);
8. (EIBEN et al., 2007);
9. (KARAFOTIAS; SMIT; EIBEN, 2012);
10. (KARAFOTIAS; HOOGENDOORN; EIBEN, 2015a);
11. (KARAFOTIAS; EIBEN; HOOGENDOORN, 2014);
12. (KARAFOTIAS; HOOGENDOORN; WEEL, 2014);
13. (LEUNG; YUEN; CHOW, 2012);
14. (MATURANA; SAUBION, 2008);
15. (ROST; PETROVA; BUZDALOVA, 2016);

Table 4 – Grouping studies by control methods and controlled parameters.

	Mutation Rate	Mutation and Crossover Rates	Others	All Parameters
Reinforcement Learning	15, 9			10, 11, 12, 8
Predictive Parameter Control		6, 5		2
Others	7	4, 3, 14, 1	13	

Source: Produced by the author.

3.2.2.1 Reinforcement Learning

This group includes studies that applied RL algorithms to implement a parameter controller.

In 2006, Eiben *et al.* (EIBEN et al., 2007) combined the RL algorithms Q-Learning and SARSA to build an out-of-the-box parameter controller. To the best of our knowledge, this was the first time an RL algorithm was used to design an out-of-the-box parameter controller for EA and SI, even though the authors did not claim that the proposed approach could be applied to multiple scenarios. In this approach, the Q function is approximated by a regression tree. The action is chosen by a GA that optimizes such an approximated function. The approach was tested on 10 multimodal randomly generated functions. The results of the experiments showed that the proposed approach overcame a standard GA in the most complex functions. However, such a success came with an overhead in the optimization process: for each action to be taken, a GA must be fully executed.

In 2012, Karafotias *et al.* (KARAFOTIAS; SMIT; EIBEN, 2012) talked about out-of-the-box parameter controllers for the first time, proposing a new RL-based approach. In this work, the authors have defined the components of any technique that is intended to be built within their framework:

- A set of observed variables extracted from the population of the evolutionary or swarm-based algorithm;
- A set of actions that are converted into parameter values;
- Any function that maps the vector of observed variables to a vector of parameter values can be used as policy.

The authors have also divided parameter control methods into two classes: static, which outputs the same parameter values for the same inputs; and dynamic, which might output different values for the same input. A pipeline with 4 different stages for the computation of the observed variables was defined as depicted below (not every stage must be visited):

1. *Source*: The current state of the optimization algorithm is represented with its raw data, *e.g.* the positions of each candidate solution, their velocities, fitness values, etc;
2. *Digest*: a function maps such a raw data to a given value v that summarizes the state of the algorithm from a certain point of view, *e.g.* best fitness, population diversity, etc;
3. *Derivative*: the variation of v from the previous iteration to the current one is calculated; and finally;
4. *History*: stores the transformed v's from the last stage from previous iterations;

For the experiments, a $(10 + \lambda)$-ES with gaussian mutation, uniform random parent selection and no recombination was used. Its mutation step size was the controlled parameter. The authors defined 4 observed variables. One of them is a *history* vector of a fitness-related digest observed variable. The other ones are single variables related to the fitnesses of the population and its diversity. Different combinations of these observed variables were tested. The policy was approximated by an fully connected ANN without hidden layers, which was trained in an off-line phase in order to define the parameter control policy to be used afterwards. The results were quite promising, since the algorithm overcame tuned versions of the original ES algorithm in many cases and the self-adaptive ES versions implemented with specific heuristics for the mutation step-size adaptation. The authors state that the choice of the set of observed variables, the set of parameters to be controlled and the algorithm to approximate the policy is critical to the success of the proposed framework.

In 2014, Karafotias *et al.* proposed an out-of-the-box parameter controller based on RL using Temporal Difference with eligibility traces (KARAFOTIAS; HOOGENDOORN; WEEL, 2014). In this paper, the authors used genotypic and phenotypic diversities, fitness standard deviation, fitness improvement and a stagnation counter as observed variables. The reward was calculated by the ratio between the variation of the best fitness during the last run of the metaheuristic and the number of objective function evaluations needed to achieve such a performance. The set of actions was composed by all the possible combinations of values of the controlled parameters. For discrete parameters, the number of possible values is finite. However, for continuous parameters, the authors decided to discretize their previously defined ranges. Thus, for such parameters, one of the intervals created by the discretization process is selected when an action is taken. Then, its final value is chosen through interpolation or probabilistic sampling. Both ways were tested in the experiments.

In this approach, the state of the metaheuristic is inferred from the observed variables through a binary decision tree that is built during the execution of the algorithm, according to the history of actions, rewards and observed variables. Each node of this tree represents an observed variable and a cutting point, while each leaf represents one of the discrete states of the environment (*i.e.* metaheuristic). The tree starts with a single leaf that represents the only state known by the algorithm in the beginning of the learning process. As the process goes on, new observed variables and rewards are computed and passed through the tree. The path along the branches is defined according to the values of the observed variables and the sequence of encountered nodes. When this tree walk reaches one of the leaves, a state is assigned to the given set of observed variables, which is "stored" in the given leaf. Whenever a leaf presents a set of observed variables that can be segmented into two clearly disjointed groups, the leaf will be turned into a decision node and two new child leaves are created. Thus, the state space is gradually segmented over time.

The authors applied the proposed algorithm to four state-of-the-art EAs solving benchmark optimization problems. All parameters for each algorithm have been controlled, including the population size. The results showed that, for the cases where there is room for improvement in the performance of the optimization algorithm solving a given problem, the proposed control mechanism obtained good results when compared to other techniques, to a random control method and to the static versions of the metaheuristics. The results also showed that the random controller presented a surprisingly good performance, what made the authors conclude that randomness seems to be an important factor for parameter control. The main drawback of this approach is the large amount of hyper-parameters needed to be defined. Another drawback is that the state tree usually gets very large. When this happens, the number of states explodes and the state-action table becomes too big. Large state-action matrices are hard to approximate, since many of the state-action pairs are never visited and their values are never updated. This behavior might cause bias overestimation on a small set of states and actions. In the same year, the same authors proposed a modified version of this approach, where the final value of the parameters are randomly selected via a uniform distribution (KARAFOTIAS; EIBEN; HOOGENDOORN, 2014).

In 2015, Karafotias *et al.* investigated the effects of using four different reward functions in their previously published approaches (KARAFOTIAS; EIBEN; HOOGENDOORN, 2014)(KARAFOTIAS; HOOGENDOORN; WEEL, 2014): 1) Improvement of the best fitness over the needed amount of time, as originally proposed; 2) A binary function, which returns 1 when there is an improvement on the best fitness, and returns 0 otherwise; 3) The difference between the current value returned by the formula used in the reward calculation number 1 and the average between the last non-zero values; 4) The raw value of the fittest candidate solution (KARAFOTIAS; HOOGENDOORN; EIBEN, 2015a). The experimental setup remained the same. Surprisingly, the binary function overcame the other approaches. It is important to highlight that such a reward function does not present any scale sensitivity to the objective function of the optimization problem, which means that it presents more generality.

In 2016, Rost *et al.* proposed two methods to overcome the limitations of the approach proposed by Karafotias *et al.* in 2014 (KARAFOTIAS; EIBEN; HOOGENDOORN, 2014)(KARAFOTIAS; HOOGENDOORN; WEEL, 2014):

1. Application of an entropy-based adaptive range parameter control proposed by Aleti and Moser (ALETI; MOSER, 2013), which will be detailed later, so that the number of bins should not be defined by the user (*i.e.* the user would not have to be concerned about the granularity of the action space);

2. Use one agent for each parameter, where the action space is segmented following the ideas of state space segmentation used in (KARAFOTIAS; EIBEN; HOOGENDOORN, 2014).

In the experiments, the authors have decided to control only the mutation step-size. For experimental purposes, a single EA was used, which solves a set of benchmark mathematical functions. The results showed that the approach number 2 is more sample-efficient. The drawback of this approach is that using a single isolated agent for each parameter may not capture any dependency between these parameters. However, this downside is not properly explored in this paper, since only one parameter is controlled in the experiments (ROST; PETROVA; BUZDALOVA, 2016).

3.2.2.2 Predictive Parameter Control

This group includes the studies that used time-series concepts to infer the probability of success for each value of each parameter at each point of time.

In 2011, Aleti and Moser proposed an algorithm that, using past performances, approximates the probability distribution that determines how likely each value of the parameters will produce an optimal outcome for each cycle (ALETI; MOSER, 2011). Besides, the algorithm is also able to chose one variation among a set of variations of a given operator.

The distributions are calculated using a method called Predictive Parameter Control (PPC), which uses the least squares method to approximate the probability of success for a given value of a given parameter at a given time. It is important to mention that these probabilities are assigned to an interval of values with fixed size instead of a single value, from which the final parameter value is sampled. For this study, the authors assumed that the values to be predicted vary linearly over time.

The Royal Road and The Component Deployment Problems were chosen to be solved by two very different EAs. The crossover and mutation probabilities were controlled by the PPC method. A clear downside of the proposed approach is that it does not take into account any information about the population state itself. It considers only a time-related variable and the previous performances of each possible value for each parameter, besides the success rate for a each parameter values. Another downside is the use of linear regression to predict the optimal parameter values, what imposes serious limitations to the method given the complexity of the problem. Even though the proposed algorithm outperforms other parameter control approaches, the authors stated that a more complete experimental benchmark should be used in a future work.

In 2012, Aleti *et al.* proposed an improvement for their previous work (ALETI; MOSER; MOSTAGHIM, 2012). Instead of sampling the parameter values from intervals with fixed size, the sizes of the intervals are adapted through time. In such a mechanism, the most successful range is divided into two equally-sized ranges. Then, these ranges inherit the probability of being chosen from the original range. The worst performing range is merged with the worst range among its neighbors. The probability of choosing the merged range is equal to the highest probability between both original ranges. This probability of choice of a range is calculated by the ratio between the number of times the choice of this range produced an improvement on the performance of the metaheuristic and the number of times the given range was chosen. This method increases the exploitation on the most promising ranges and the exploration on the worst performing ones.

For the experiments, the authors applied the new parameter control method to EAs solving the Royal Road, the Generalized Quadratic Assignment and the Component Deployment problems. The crossover and mutation rates were chosen to be the controlled parameters. The reward is calculated based on the fitness values returned by the objective function. Since the Component Deployment Problem is multi-objective, a hypervolume-based indicator is used for this purpose. The results of the experiments showed that the proposed technique outperforms all state-of-the-art algorithms up to the date of the publication, especially for the most complex problems. The authors claim that the superiority of the proposed method is due to the fact that the most promising areas of the parameter space are quickly and deeply exploited. However, high-performing ranges are sometimes merged to large intervals, turning it difficult to reestablish the exploitation behavior on that region. Another drawback is that, as its predecessor, the parameter adjustment is still being made considering only time-related variables.

In 2013, Aleti and Moser proposed another improvement on their previous studies (ALETI; MOSER, 2013). In this work, the K-Means algorithm (Lloyd, 1982) guided by information entropy is used to cluster parameter values according to their contribution to the performance of the metaheuristic. The formed clusters define the boundaries between bins for each parameter. The same experimental setup developed in the previous work was used in this one, except for the exclusion of the Royal Road problem. The new approach consistently outperformed all other state-of-the-art techniques on the experiments, including its predecessor.

In 2014, Aleti *et al.* investigated four different time-series prediction techniques to be used with their parameter control technique proposed in 2011 (ALETI et al., 2014):

- Linear regression, as published in their original paper;
- Simple Moving Average, where the average value of past performances is used to predict future responses, which assumes that the historical trend continues;
- Exponentially Weighted Moving Average, where the average of past data points is calculated with exponentially increasing weights assigned to each point.In this approach more importance is given to the most recent executions, which makes the same assumptions as the technique number 2, but presents higher sensitivity to noise;
- Autoregressive Integrated Moving Average (ARIMA) (NEWBOLD, 1983), which is a combination of linear and moving average techniques.

These prediction methods make statistical assumptions about the data points that must be verified before using them. Therefore, the authors analyzed such statistical properties on data collected from previous executions of EAs for each of the following parameters: mutation rate, crossover rate, population size, mating pool, and variations of some of these operators. It was concluded that all prediction techniques are satisfactorily suitable for all of them, except for population size, which presented extra difficulties. Regarding the experimental results, the linear regression predictor showed the best performance. The experiments with population size control showed that, even though no prediction method is suitable to predict the quality of its values, controlling the population size did not harm the performance of the original underlying EA.

3.2.2.3 Others

This section groups up the studies that used other approaches than RL or PPC to create a parameter controller.

Aine *et al.*, 2006, proposed a parameter control method based on Dynamic Programming (AINE; KUMAR; CHAKRABARTI, 2006). In this approach, the parameter control problem for EAs was modeled as an optimization problem where the expected quality of the population of an EA in a given state at a certain point of time is maximized by choosing the optimal set of parameter values and setting a certain amount of time to run the algorithm. The authors authors used only population diversity and quality measures to infer the state of the metaheuristic. They claim that their approach is able to consider time constraints and adapt its choices to fit to them.

As a proof of concept, the authors controlled the crossover and mutation rates of a GA solving random instances of the Traveling Salesman Problem. The authors compared the performance of the proposed algorithm to a tuning method. The results showed that, under the posed time constraints, the dynamic approach achieved better results than the static method. However, the proposed technique was tested on a single problem, even though it was applied to many different instances of it. Given the very limited set of observed variables defined by the authors, there is no guarantee that such an approach would be successfully applied to other problems and/or metaheuristics.

In 2008, Maturana and Saubion (MATURANA; SAUBION, 2008) proposed a parameter control method based on fuzzy logic. Their approach was divided into two phases: the learning and the control phases.

During the learning phase, the effects of different parameter combinations on the diversity and quality of the population are learned from previous executions of the optimization algorithm. The function that maps such variables to one another is approximated by a Takagi-Sugeno Fuzzy Logic Controller with polynomials of order 1. The coefficients of these polynomials are defined through multivariate linear regression. On the control phase, a heuristic is used to control the diversity of the population, using as feedback a fitness-based numerical signal and the diversity itself. The parameter values are set through the function that maps such a diversity to values that must be used in order to reach such a performance.

In this paper, the authors have tested three different control heuristics. A permutation-encoded GA was used to solve 38 instances of the Quadratic Assignment Problem. The results were compared only between the proposed heuristics. The main drawback of this approach is that if a large set of observed variables is used, the fuzzy functions becomes prohibitively complex.

In 2011, Chatzinikolaou presented a self-adaptive agent-based, peer-to-peer GA for parameter control (CHATZINIKOLAOU, 2011). In this proposal, a multi-agent system is implemented, where each agent runs an independent GA and, from time to time, they interact with each other. In this interaction, they exchange candidate solutions and parameter values. The communication between agents is made through a protocol called Lightweight Coordination Calculus (ROBERTSON, 2005). The multi-agent system works as depicted below:

1. Each agent runs its optimization process for some time and computes its average fitness. The average fitness defines the quality of each agent.

2. Each agent shares its performance with its neighbor agents. Then, it chooses with whom it will exchange candidate solutions and parameters via a roulette wheel selection.

3. The recombination of candidate solutions between agents is done by exchanging candidate solutions, while the new parameters are obtained by averaging the parameter values of both agents and applying a random mutation to them.

The author highlights that the agents are executed asynchronously, in such a way that the system is able to keep running even with communication issues between agents or some of them are down (what obviously harm the overall performance of the algorithm). In fact, the system is able to work even with a single agent, even though, in this case, the parameter set will remain constant during the whole optimization process. The main advantage of the proposed method is that not only the parameters control can benefit from the collective behavior of the population of agents, but it is also highly scalable in a distributed platform. It is important to note that, even though the author presented this algorithm as a metaheuristic itself, the parameter control proposed in this work can be used with any other metaheuristic and parameter.

Despite the very interesting proposed mechanism, the authors applied their technique to only one benchmark function. Also, they did not compare their proposal to any other. Instead, this work has intended to evaluate the influence of the information exchange and the size of the population of agents on the overall performance and on the evolution of the mutation rate. Thus, the author tested three different variations: no communication between agents; exchanging only candidate solutions; exchanging candidate solutions and parameters. Also, different agent population sizes were tested. The best results were achieved with full communication between agents, where candidate solutions and parameters are exchanged. Moreover, the larger the number of agents, the better is the performance of the algorithm.

In 2012, Leung *et al.* proposed a parameter-less out-of-the-box parameter control called Parameter Control System Using Entire Search History (PCSH) (LEUNG; YUEN; CHOW, 2012). The PCSH algorithm uses previous solutions generated by the underlying metaheuristic and their fitnesses to approximate the objective function's landscape. With this information, the proposed algorithm can choose parameter values that maximize the approximated objective function. A Binary Space Partitioning Tree is used to build the surrogate model. The authors argued that this algorithm was chosen mainly because the it is parameter-less.

In order to evaluate the proposed method, 3 metaheuristics were used: GA, DE and PSO. The following parameters were controlled: Crossover operator, crossover values and mutation operator (GA); crossover operator, crossover values and differential amplification factor (DE); and the cognitive, social and inertia weights (PSO). A set of more than 30 well-known benchmark optimization functions was used to evaluate the proposed technique. The parameter-less aspect of the algorithm can be considered as an advantage, since it completely removes the need of parameter adjustment. However, it may cause some difficulty for the algorithm to generalize to multiple classes of problems. Another advantage is that, according to the authors, using a surrogate model to evaluate the approximated quality of a parameter setting saves a lot of computational resources. Also, this model might be increasingly improved as more and more data is generated during the optimization tasks. The presented results are quite interesting and encouraging.

In 2013, Bielza *et al.* proposed a parameter control algorithm using Bayesian Networks, where each node is a parameter of a GA (BIELZA; POZO; LARRAñAGA, 2013). This makes the algorithm able to learn the correlation between parameters. The Bayesian network is induced from a training dataset composed by multiple runs from the metaheuristic. These runs are executed during a reduced number of generations with a small population in order to reduce the computational burden of the process. The parameters of the GA are set according to the joint distribution function encoded by the learned network. Even though the authors present such an approach as an improved GA, the use of Bayesian Networks for parameter control as proposed in the paper can be clearly extended to any algorithm or parameter.

The authors applied their algorithm to the problem of optimal ordering of tables. These were the controlled parameters: population initialization method, crossover operator, crossover rate, mutation operator, mutation rate, selection method and stopping criteria. The experiments showed that the new approach presented very similar results to state-of-the-art algorithms for the problem at hand. However, it dramatically reduced the computational burden needed to achieve such a performance. Also, for the largest instances of the problem, the proposed algorithm overcame all other approaches.

3.3 CONCLUSIONS

After presenting an overview of the results of this literature review and detailing every selected paper, here the four research questions previously posed for this literature review are answered.

RQ1: Which methods for out-of-the-box parameter control have been used in the literature?

We have found that 66.67% of the selected articles used RL or time-series analysis methods to create an out-of-the-box parameter control method. The main disadvantages of the RL-based approaches analysed in this review are the large set of hyperparameters to be adjusted. Also, the tree-based approximation method for state definition may generate very large tress, what might incur in the problems already mentioned in this study. The time-series prediction-based approaches provide smaller sets of hyperparameters, but use only time-related variables, hence ignoring any information that could be extracted from the population. This characteristic gives less approximation power to the models used in these controllers when compared to the RL-based ones. The remaining papers presented various different solutions, such as DP, fuzzy logic, agent-based GA, surrogate models and Bayesian networks. Since there is only one approach proposed for each of these methods, we cannot summarize their characteristics as a single group, as we did for RL and PPC-based methods. It is also important to highlight that 4 of the 5 last published papers, which corresponds to the latest 6 years of publications, applied RL on parameter control, which produced interesting results.

RQ2: Which methods for out-of-the-box parameter control are self-adaptive on their own parameters?

Unfortunately, only one of the accepted studies proposed a parameter-less control method: Leung *et al.*, 2012 (LEUNG; YUEN; CHOW, 2012).

RQ3: Which methods adapt its search to the limitations of the underlying hardware, time budget, or maximum number of fitness evaluations?

None of the accepted studies were concerned with the extra computational burden added by the control method to the controlled algorithm.

RQ4: What are the main challenges and future trends for out-of-the-box parameter control for EA and SI?

The problem of parameter control is far from being satisfactorily solved. Regarding out-of-the-box parameter control methods, things become even more difficult. Here are the main challenges that we have identified after concluding the current literature review:

1. Training EA and SI parameter control policies with RL can be very computational-demanding. And to date no study has ever proposed a scalable approach that could benefit from parallel and distributed computing platforms.

2. RL algorithms usually require the adjustment of many hyperparameters, what makes difficult its successful use. Also, the search for an optimal policy can be very unstable, since RL algorithms usually suffer from bias overestimation caused by the "dog chasing its tail" effect in the Bellman Equations. It means that, for many algorithms,

the policy search is likely to exploit wrong decisions and stuck in local minima (FUJIMOTO; HOOF; MEGER, 2018).

3. Even though the authors of the reviewed studies argue that they propose out-of-the-box methods, very limited benchmarks have been used to assess such a generality, what reduces the scope and generality of these methods.

4. To the best of our knowledge, no one has explored the transferability between different problems, in a way that an optimal policy is chosen among many policies trained over a set of training functions, aiming to maximize the performance of the metaheuristic in an unseen testing function.

4 TRAINING PARAMETER CONTROLLERS WITH DISTRIBUTED REINFORCEMENT LEARNING

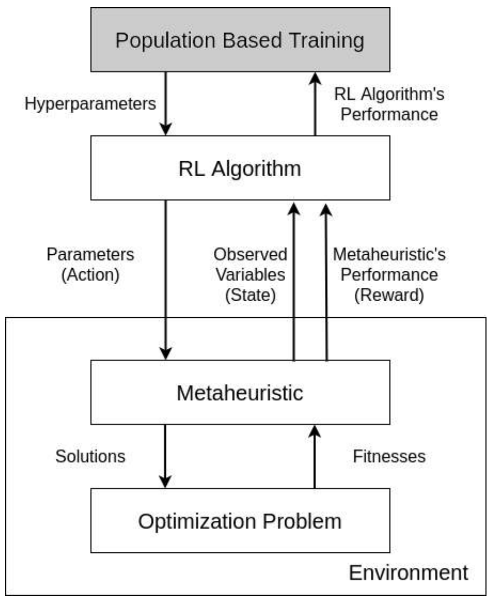
"Everything you can imagine is real."

Pablo Picasso

In this thesis, we put forward an out-of-the-box training methodology for parameter control policy for EA and swarm-based algorithms with distributed Reinforcement Learning. We suggest that in the training process, the PBT algorithm is used to evolve the parameters and hyperparameters of the RL algorithm, which then controls the parameters of a metaheuristic that solves an optimization problem. Figure 9 shows the general scheme of the proposed training approach. It can be seen that the running metaheuristic and the optimization problem form the environment that the RL algorithm interacts with. The RL algorithm acts on the environment by setting the parameters of the metaheuristic. Then, the metaheuristic runs for one iteration, changes its state, and returns it to the RL algorithm in the format of observed numerical variables. The state of the environment is returned alongside a performance measure (*i.e.* reward).

As already mentioned, the objective of the proposed method is to train a policy for a given metaheuristic running on a set of training functions and successfully apply the trained policy on an unseen testing function, limited by a predefined budget. The idea is to perform one training process for each training function, and for each finished training

Figure 9 – General scheme of the proposed training method on the RL framework.


Population Based Training
Hyperparameters
RL Algorithm's Performance
RL Algorithm
Parameters (Action)
Observed Variables (State)
Metaheuristic's Performance (Reward)
Metaheuristic
Solutions
Fitnesses
Optimization Problem
Environment

Source: Produced by the author.

epoch, the most up-to-date policy is stored in a pool of trained policies. Then, one of the trained policies in the pool is chosen to be applied to the unseen problem. In any problem where machine learning algorithms are used to solve it, the training data must be chosen in an attempt to reproduce the distribution of the production data (*i.e.* test data or unseen data). Likewise, in our method, the set of training functions must correspond to problems of which the objective functions share some characteristics with the unseen problem. The reason for this is that similar problems may generate similarly distributed data during the search process of the metaheuristic. Thus, the success of the proposed method depends on the choice of such a training set. However, the identification of similar problems depends on the previous knowledge of the users about their objective functions.

The policy training process is key for the success of the proposed approach. This process is described next.

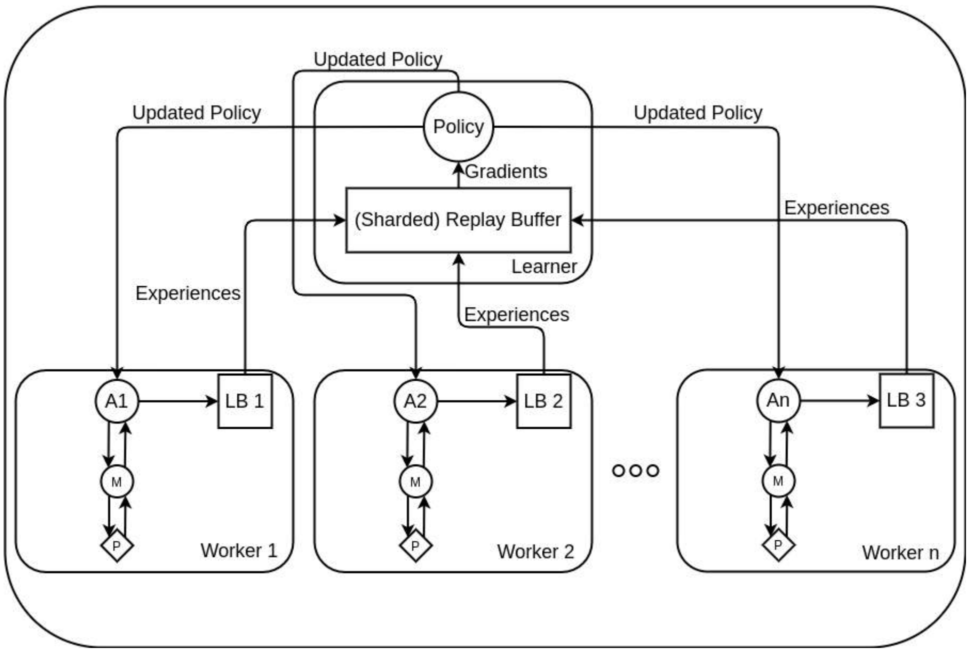
4.1 LEARNING THE POLICIES

In our method, multiple workers interact with multiple instances of the environment collecting experience data in parallel, one copy of the environment for each RL worker. Figure 10 shows multiple RL workers running in a distributed arrangement, where A1, A2, and An represent their local policies, M is the controlled metaheuristic, P is the optimization problem and LB is its local buffer. The interaction between workers and their instances of the environment is made through the observation of the state variables that describe the environment state and the actions taken accordingly by copies of the current policy. These state variables are described later in this section. A representation of such a worker, which we will call RL worker from now on, can be seen in Figure 11.

Algorithm 7 shows the pseudocode of the RL worker. Each RL worker runs independently collecting experiences and storing in their respective local buffers until they are completely full. Then, the collected experiences are copied into the centralized replay buffer. If the workers run asynchronously, the replay buffer is divided into parts, one for each worker. Otherwise, the replay buffer works as a single shared memory. Whenever this memory becomes full of transitions, the oldest ones must be removed. After finishing the replay buffer update, each worker cleans up its local buffer and requests an update of its local policy from the centralized learner. The centralized learner then responds as soon as it is ready to do so.

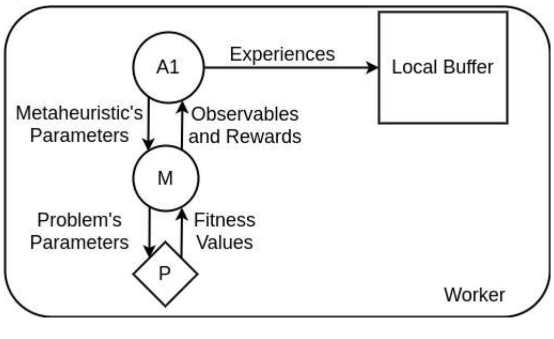
Algorithm 8 shows the pseudocode of the centralized learner, where the new policy is learned. When the workers run asynchronously, the learner thread updates the policy parameters asynchronously as well. In this case, it updates its parameters continuously, without waiting for the workers to finish updating the replay buffer. On the other hand, when these workers run synchronously, the learner updates its policy parameters whenever all workers have finished copying their transitions into the replay buffer. Line 3 in Algorithm 8 works as a synchronization barrier for such cases. It can be seen that this

Figure 10 – Training of the parameter controller using distributed Reinforcement Learning (single PBT worker). Multiple RL workers work in parallel inside a PBT worker to collect experiences and transfer them to the centralized replay buffer. The centralized learner adjusts its current policy based on these collected experiences. After “uploading" its collected experiences, each worker requests an update of its local policy. The RL workers run synchronously or asynchronously.



Source: Produced by the author.

Figure 11 – Single RL worker in the training process.



Source: Produced by the author.

Algorithm 7: Pseudocode of the RL worker.

```
Input: Most up-to-date parameter control policy.
Output: Set of collected experiences.
1  REQUEST from the centralized learner an update of its local policy
2  CLEAN the local buffer if it is full of previous experiences
3  while Local buffer is not full do
4      INITIALIZE the environment (i.e. initialize the metaheuristic and the
         optimization problem)
5      CALCULATE the current values of the state variables of the environment
6      while the budget of the metaheuristic is not exhausted do
7          TAKE an action by defining the current parameter values of the
             metaheuristic according to the current values of the state variables
8          SET the new parameter values of the metaheuristic
9          RUN the metaheuristic for one iteration
10         GET the reward for the taken action
11         CALCULATE new values for the state variables of the environment
12         STORE the n-tuple (previous values of state variables, action, reward,
             new values of state variables) in the local buffer as a new experience
13 return Set of collected experiences.
```

mechanism embraces both synchronous and asynchronous distributed implementations of RL algorithms, such as TD3 and Ape-X DDPG, respectively.

After updating the current policy, a testing phase is executed. At this point, the most up-to-date policy is tested on the remaining training functions, so that the current policy can be stored at the pool of trained policies alongside its performance on such functions. The testing and storage phases are represented in lines 5 and 6.

Algorithm 8: Pseudocode of the centralized learned running mechanism.

```
Input: Initial policy.
Output: Trained policy.
1  while Stopping criteria are not met do
2      if Workers run synchronously then
3          WAIT until all RL workers have updated the centralized buffer
4      UPDATE the parameters of the policy
5      TEST the current policy on the set of remaining training functions
6      STORE the current policy in the pool of trained policies alongside its
         performance in the tests
7      RESPOND to all pending requests for a policy update sent by the RL workers
8  return Trained policy.
```

In our method, the PBT algorithm is used to evolve the parameters and hyperparameters of the RL algorithm on-the-fly. The training process previously described runs inside a PBT worker, as depicted in Figure 10. As previously mentioned, the exploitation and exploration mechanisms need to be defined according to the necessity of each situation where the PBT algorithm is used. For the exploitation mechanism of our method, each worker of the bottommost quantile (*i.e.* the least performing workers) randomly chooses another PBT worker in the upmost quantile to copy its parameters and hyperparameters. Such a quantile is previously defined by the user. The performance of a PBT worker is measured based on the average reward over all steps performed during its latest training epoch. The reward for each taken action is defined later in this section. Regarding the exploration mechanism, the hyperparameters of the PBT workers are perturbed following a predefined mutation function with a given probability. If not, the continuous hyperparameters are multiplied by a random factor, which can be anything between 1.2 or 0.8, or changed to one of its adjacent values if the hyperparameter is discrete. Figure 12 shows the distributed RL algorithm running under the PBT framework. The aforementioned implementation of the exploitation and exploration mechanisms is available at *https://docs.ray.io/en/master/* (LIANG et al., 2017). The pseudocode of the PBT algorithm is shown in Algorithm 6 in the previous section.

It is important to notice that the PBT algorithm allows a diverse search for the optimal policy. This implies that training policies with RL algorithms running under the PBT framework increase the chance of finding an optimal policy and avoiding bad policies. Thus, the larger the population of PBT workers, the better (as long as the hardware supports it).

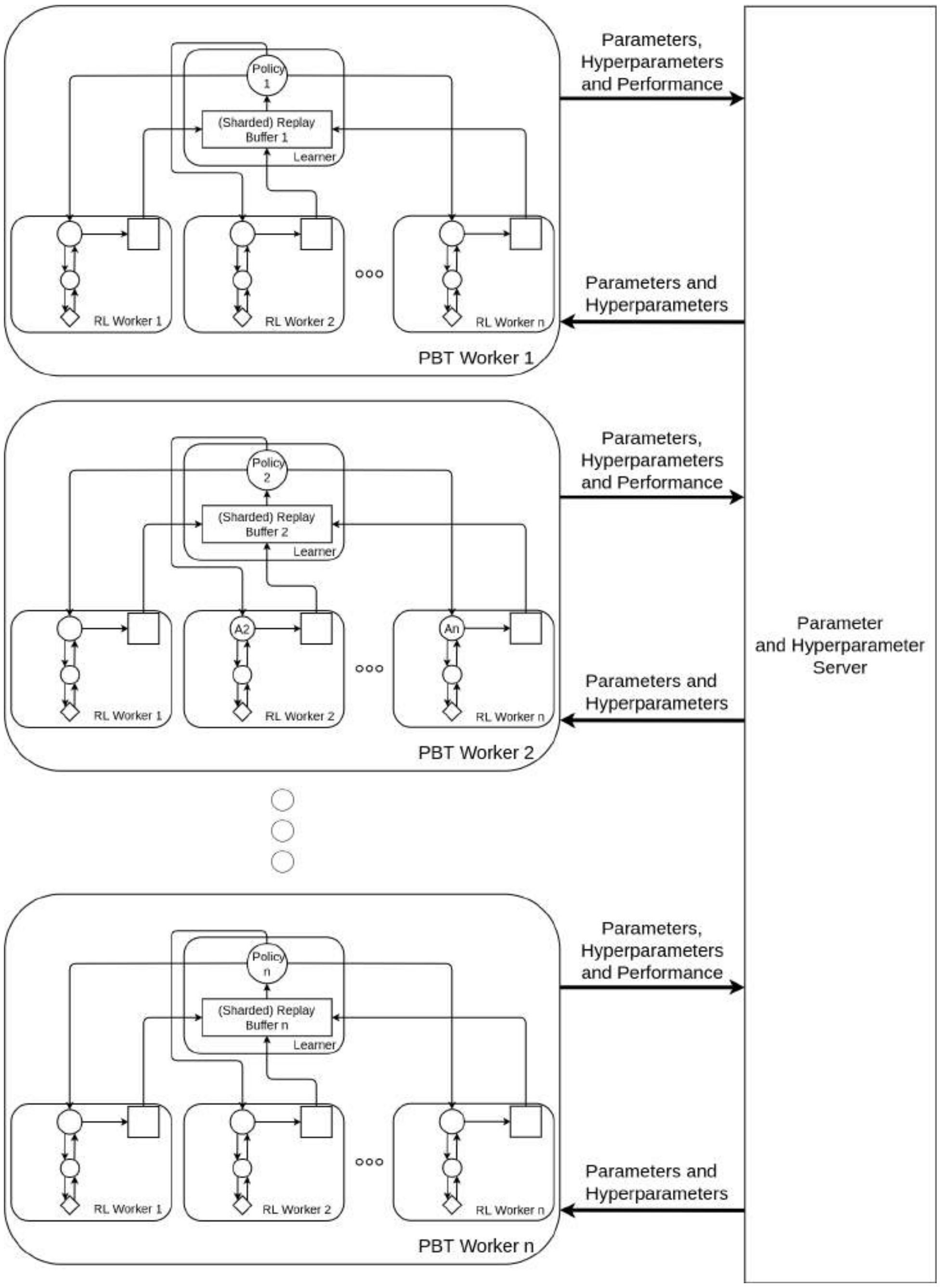
On the other hand, the number of parallel RL workers per PBT worker does not affect the capacity of exploration of the algorithm, but speeds up the training process. To understand this, it is important to notice that the size of the local buffers is the size of the centralized buffer divided by the predefined number of RL workers. Thus, the more RL workers used, the smaller each local buffer, and therefore the faster they are filled with transitions. Since the data is collected by the RL workers, until their local buffers are full, the larger the population of RL workers, the faster the centralized replay buffer is updated and, thus, the faster the training process.

It is clear that setting the numbers of RL workers and PBT workers means dealing with conflicting objectives, since any hardware has limited resources. Therefore, the user must deal with the trade-off between exploration capability and time-efficiency.

The PBT algorithm has its own hyperparameters, namely: perturbation interval (*i.e.* the interval between each exploitation/exploration execution), quantile fraction (*i.e.* the parameter/hyperparameter exchanging quantile), and resampling probability (*i.e.* the probability of resampling from the original distribution). The higher the quantile fraction, the stronger is the exploitation in the search for the policy, since more PBT workers exchange information between them. The higher the resampling probability, the stronger the exploration, since resampling from the original distribution means resetting the values of the hyperparameters.

Regarding the perturbation interval, understanding its effects is not so straightforward,

Figure 12 – Training of the parameter controller using distributed Reinforcement Learning and Population Based Training (multiple PBT workers). Multiple PBT workers work in parallel and asynchronously. They communicate with each other through the parameter and hyperparameter server, where parameters and hyparameters are exchanged between PBT workers.


Parameters, Hyperparameters and Performance
Policy 1
(Sharded) Replay Buffer 1
Learner
RL Worker 1
RL Worker 2
RL Worker n
Parameters and Hyperparameters
PBT Worker 1
Policy 2
(Sharded) Replay Buffer 2
A2
An
PBT Worker 2
Parameter and Hyperparameter Server
Policy n
(Sharded) Replay Buffer n
PBT Worker n

Source: Produced by the author.

since every perturbation of hyperparameters (*i.e.* exploration) comes after an exchange of parameters and hyperparameters between two PBT workers (*i.e.* exploitation). Therefore, the exploration/exploitation balance strongly depends on the other two hyperparameters. The impact of the numbers of PBT and RL workers on the performance of the search process is discussed later in this paper through the analysis of the results of the experiments.

It is important to note that as PBT works as the highest layer of control, the user needs to define exact values only to its hyperparameters. It means that, regardless of the RL algorithm used, the hyperparameters that need to be manually set to exact values will be the same. For the hyperparameters of the underlying RL algorithm, flexible ranges can be defined, which makes its use easier. Moreover, the same hyperparameters must be defined regardless of the used metaheuristic, what also eases the use of the proposed methods since the user does not need to understand how multiple metaheuristics work, but only the rationale of PBT and how its hyperparameters affect its behavior.

4.2 DEFINING THE REWARD FUNCTION, THE STATE VARIABLES, AND THE ACTION SPACE

As already mentioned, for each interaction of the controller with the metaheuristic, the optimization algorithm runs for one iteration. After that, its performance is returned as a reward for the taken action. This reward is calculated according to Equation 4.1, where α_r is a factor that must scale the reward to values in the order of magnitude of 10^0, which avoids instability of the learning process of the neural networks that approximate the value functions. Moreover, a_t is the action taken at time t, s_t is the state of the metaheuristic after taking such an action, s_{t-1} is the state of the metaheuristic before taking this action and $F_{max}(s_t)$ is the maximum fitness among the candidate solutions of the metaheuristic at time t.

$$r(s_t, a_t) = \alpha_r log_{10} \frac{F_{max}(s_t)}{F_{max}(s_{t-1})} \tag{4.1}$$

After preliminary experiments, α_r was set to 1, so that the previously described condition (rewards remain in the order of magnitude of 10^0) were met. With this function, the RL algorithm aims at the maximization of the ratio $\frac{F_{max}(s_t)}{F_{max}(s_{t-1})}$ for each iteration of the optimization algorithm, which means that it must maximize the maximum fitness gain between iterations. However, if the maximum fitness decreases, the penalty (*i.e.* negative reward) will be more severe than the reward received when it increases because of the log_{10} function. Such an “imbalanced” reward policy makes the agent avoid at all cost situations where the maximum fitness is decreased. It is important to highlight that this reward function is not sparse, which makes the learning process of the value functions easier. In this context, a sparse reward function is a function that often returns zero reward.

For the calculation of the reward in minimization problems, the fitness assigned to each solution found by the metaheuristic is calculated according to Equation 4.2. In this equation, $\mathbf{x}$ is the solution vector, and $f(\mathbf{x})$ is the value of the objective function for the solution $\mathbf{x}$. The denominator of the fraction in the equation ($i.e.$ $max(f(\mathbf{x}), 10^{-20})$) avoids division by zero. If the best fitness in the population is decreased in a minimization problem from iteration $t-1$ to iteration t, the reward is positive. For a maximization problem, it needs to increase for a positive reward.

$$F(\mathbf{x}) = \frac{1}{max(f(\mathbf{x}), 10^{-20})} \tag{4.2}$$

These equations were proposed by Schuchardt *et al.* in (SCHUCHARDT; GOLKOV; CREMERS, 2019). In this work, the authors applied the algorithm Proximal Policy Optimization (PPO) to control the parameters and other aspects of different metaheuristics. It is important to mention that this work was not returned by the search engines used in the already presented systematic literature review. However, the proposed method can be classified as out-of-the-box as well, and the reward function proposed in the paper turned it essential in the development of this work. Moreover, the idea of executing one single training process for each function came up to enable the use of this reward function. With such a function, if multiple training functions are interleaved during the training process, so that each training epoch is performed for a different training function, the scale of the rewards varies throughout the training process. This difference between scales turns some of the functions more important than others when the average reward over many training epochs is calculated by PBT in order to rank the PBT workers. Thus, the same training function must be used for all training episodes.

The given reward function was chosen because the PPO algorithm is a model-free and gradient-based policy for continuous action spaces, just as our chosen RL algorithm that will be discussed later. This is the first study that applied an RL method for continuous action spaces to train parameter control policies. Later on, in this work, the choice of the RL algorithm to evaluate our method will be justified and the similarities between the chosen algorithm and PPO will become clearer.

The set of state variables observed by the RL agent was inspired by the work published by Sharm *et al.* in (SHARMA et al., 2019). The values of all variables lie between 0 and 1, such that the inputs of the policy have the same numerical importance. Table 5 provides the state variables used in this work, where f_{wsf} is the worst fitness so far, b is the current budget left, B is the initial available budget, b_e is the elapsed budget since the last improvement of the best fitness so far, p_{a_i} is an candidate solution with index a_i picked from the PBA, a_i and b_i are random indexes, max and min are the vectors with all their values set to the maximum and minimum values for each dimension of the PBA's search space, respectively, p_{bf} is the candidate solution of the PBA with the best fitness, f_{a_i} is the fitness of an candidate solution with index a_i picked from the PBA, f_{bf} is the current

Table 5 – Set of state variables.

Variable definition	**Equation**	**Length**
Normalized difference between the best fitness so far (f_{bsf}) and the average fitness of the population ($\overline{F}$).	$\frac{f_{bsf}-\overline{F}}{f_{bsf}-f_{wsf}}$	1
Ratio between the standard deviation of the witnesses of the population ($std(F)$) and the maximum standard deviation between two candidate solutions in the population.	$\frac{std(F)}{std(\{f_{bsf},f_{wsf}\})}$	1
Percentage (between 0 and 1) of the remaining budget of the episode.	$\frac{b}{B}$	1
Percentage (between 0 and 1) of the budget elapsed since the last improvement of the best fitness.	$\frac{b_e}{B}$	1
Normalized Euclidean distance between 100 pairs of randomly chosen candidate solutions.	$\frac{\lVert p_{a_i}-p_{b_i}\rVert}{\lVert max-min\rVert}, \forall i \in [0,99]$	100
Normalized Euclidean distance between the current fittest candidate solution and 100 other randomly chosen candidate solutions.	$\frac{\lVert p_{bf}-p_{a_i}\rVert}{\lVert max-min\rVert}, \forall i \in [0,9]$	10
Normalized absolute difference between the fitnesses of 100 pairs of randomly chosen candidate solutions.	$\frac{\lvert f_{a_i}-f_{b_i}\rvert}{f_{bsf}-f_{wsf}}, \forall i \in [0,99]$	100
Normalized absolute difference between the currently best fitness and 100 randomly chosen candidate solutions.	$\frac{\lvert f_{a_i}-f_{bf}\rvert}{f_{bsf}-f_{wsf}}, \forall i \in [0,9]$	10
Normalized absolute differences between the currently best fitness ever found so far and 100 randomly chosen candidate solutions.	$\frac{\lVert p_{bsf}-p_{a_i}\rVert}{\lVert max-min\rVert}, \forall i \in [0,9]$	10
Number of improving candidate solutions for each one of the latest 10 iterations.	$\frac{I_{t-k}}{N_{t-k}}, \forall k \in [0,9]$	10
Whether the best fitness ever has improved or not for each one of the latest 10 iterations.	$I_{bsf_{t-k}}, \forall k \in [0,9]$	10

Source: Produced by the author.

best fitness among all candidate solutions of the PBA, p_{bsf} is the position of the solution with the best fitness so far, I_{t-k} is the number of candidate solutions that improved their fitness during the $(t-k)^{th}$ agent's iteration (*i.e.* $k=0$ is the current t^{th} iteration), N_{t-k} is the number of candidate solutions in the PBA in the $(t-k)^{th}$ agent's iteration, and $I_{bsf_{t-k}}$ is a boolean variable that is set if the best fitness so far was improved during the $(t-k)^{th}$ agent's iteration.

It is important to mention that every random index is sampled with replacement from a uniform distribution. After concatenating all observed variables, the vector of observations has 254 floating-point values.

As already mentioned, the agent works in a continuous action space. Thus, its actions are continuous values assigned to each parameter of the metaheuristic. In our approach, these actions are values between 0 and 1, which are rescaled to predefined parameter ranges through Equation 4.3, where p_i is the original value returned by the policy to the parameter i, $p_{i,max}$ is the upper bound of the range of values for parameter i, $p_{i,min}$ is the lower bound of the same range, and p'_i is the final value set to the parameter i.

The decision to keep the policy output between 0 and 1 was made to avoid an unstable behavior of the the neural networks that approximate such a policy in the execution of the learning algorithm.

$$p'_i = p_i(p_{i,max} - p_{i,min}) + p_{i,min}. \tag{4.3}$$

4.3 CHOOSING A TRAINED POLICY

As previously mentioned, a pool of trained policies is generated with a set of training problems during the training phase. After that, a policy must be chosen in order to solve an unseen problem. Such a choice aims at the maximization of the performance of the chosen metaheuristic on the unseen problem. This mechanism is proposed and described in this section.

Let P be a set of trained policies and T a set of trained functions. Each policy in P is generated after a training epoch of PBT worker during the training process with a given metaheuristic m optimizing a given objective function $f \in T$. It is important to notice that, since each training epoch of each worker generates one new policy, there are multiple policies trained with each function $f \in T$.

In order to find a policy that maximizes the performance of the metaheuristic m on the unseen objective function f_u, a score must be calculated for each policy $p \in P$. This score is calculated by testing p for each training function $g \in T - \{f\}$, where f is the objective function used to train p. Then, the performance of each policy p for each function $g \in T - \{f\}$ is defined as $\frac{n(P^{<}_{p,g})}{n(P-\{p\})}$. In this formula, $n(P^{<}_{p,g})$ is the number of policies in P that presented a lower average performance over 30 runs than p on the function g, and $n(P - \{p\})$ is the total number of trained policies in P except p. It means that the performance of p in g is the percentile of the average performance of p running 30 times on g (*e.g.* fitness, in case g is a maximization problem) among all policies in P running on g. After calculating the scores of the policy p for each function g, the final score is calculated by averaging them. The policy with the highest score is chosen to be used on the unseen objective function f_u.

It is important to note that the choice of the best policy for a given unseen function f_u is not guaranteed, since this choice is made through the observation of the performances of p in different functions $g \neq f_u$. The results achieved by the such a policy selection method can be seen in Section 5.2.3, where the generality of the proposed method is assessed.

5 EXPERIMENTAL RESULTS AND DISCUSSION

"What is research but a blind date with knowledge?"

Will Harvey

The experiments in this study have been designed to verify whether the proposed method is able to address the three gaps found in the literature presented in the early chapters of this work or not. Such gaps are summarized as follows:

1. Training EA and SI parameter control policies with RL can be very computational-demanding.
2. RL algorithms usually require the adjustment of many hyperparameters, what makes difficult its successful use. Also, the search for an optimal policy can be very unstable.
3. Very limited benchmarks have been used to assess the generality of the out-of-the-box methods proposed so far in the literature.

This chapter is divided into two parts: Experimental Methodology, where the methodology used in this work to perform the experiments is provided, and Results and Discussion, where the experimental results are delivered.

5.1 EXPERIMENTAL METHODOLOGY

5.1.1 Choosing the Reinforcement Learning Method

A few aspects must be taken into account to choose the right RL algorithm to solve a given problem. First of all, it is necessary to choose between model-based and model-free algorithms. Model-based algorithms create an internal model of its surrounding environment. Even though good models give to the agent the capability of accurate long-term planning, building accurate models can be very hard or even infeasible in real-world applications. A successful example of a model-based RL algorithm is AlphaZero (SILVER et al., 2017).

Model-free algorithms work by making decisions without any information about the state of the environment and transition probabilities. As previously mentioned in this work, model-free methods learn by experience. Even though these algorithms are less sample-efficient, they are easier to implement and to make them work, since they do not depend on the quality of the model of the environment. Thus, in this work, only model-free algorithms are considered.

As previously mentioned, model-free algorithms can be divided into two subgroups: policy optimization methods and Q-learning-based methods. Policy optimization methods are usually more stable. However, Q-Learning algorithms are usually more sample-efficient. Given this trade-off, intermediate methods, which mix the characteristics of both classes of algorithms, are worth being considered. As presented in the background chapter of this work, TD3 and Ape-X DDPG are two of the state-of-the-art intermediate algorithms. However, TD3 was proposed as an improvement of the canonical DDPG, promising more stability in the learning process. Even though the asynchronous workers of Ape-X DDPG would benefit more from a parallel platform, its usual instability, *i.e.* its higher probability of exploiting bad actions, does not pay off in our problem, as observed in preliminary experiments. Therefore, in this work, we used TD3 to train the parameter control policies.

5.1.2 Setting up the Hyperparameters of TD3 and PBT

The implementation of TD3 used in this paper are available in the module *rllib* of a Python library called *Ray* (LIANG et al., 2017). Some of its hyperparameters were controlled by the PBT algorithm, while others were kept fixed.

The constant hyperparameters of the TD3 algorithm used in the experiments of this study were defined as follows. The algorithm Adam was used as the parameter optimizer of the neural networks (KINGMA; BA, 2014).

- Update delay between policy and Q-Function parameters: 2 (*i.e.* for each policy update, the Q-Function is updated twice);
- Target noise (*i.e.* variance of the gaussian noise ϵ): 0.2;
- Target noise clip (*i.e.* c): 0.5;
- Standard deviation of the zero-mean gaussian noise added to the actions: 0.1;
- γ: 0.99;
- Initial random steps (*i.e.* number of steps with random decisions executed before the algorithm starts learning): 45000;
- Adam β_1: 0.9;
- Adam β_2: 0.999;
- Adam ϵ: 10^{-7};

The frequency in which the target network is updated is also constant for each of the PBT workers. However, each PBT worker sets a different value to such a hyperparameter.

To set these values, considering that 16 PBT workers are used, the interval [100, 5000] is divided into 15 equally sized intervals. These 16 equally distanced values are used as the target network update frequency for the PBT workers, one value for each worker. Regarding the dynamic hyperparameters of TD3 controlled by the PBT algorithm, $\alpha_{\boldsymbol{\theta}}$ and $\alpha_{\boldsymbol{\phi}}$ are initially randomly set to a number within the interval [10^{-6}, 10^{-1}] following a log uniform distribution, while τ and the L2 regularization hyperparameter are randomly chosen within the ranges $[10^{-4}, 10^{-2}]$ and [0,1] respectively, these ones using a uniform distribution.

The number of hidden layers of the neural networks used in both algorithms was set to 4 with 300 neurons each. Relu was used as the activation function for all neurons in the hidden layers. For each interaction between the RL worker policy and the corresponding environment, one iteration of the metaheuristic solving a given optimization problem is executed. Each episode is composed of 45000 of these iterations, what generates 45000 transitions. Since one epoch was composed of 300 iterations of the metaheuristic, each episode is composed of 150 of these epochs. The size of the centralized replay buffer was set 45000. As already mentioned, the size of the local buffers was defined as the size of the centralized buffer divided by the number of PBT workers.

Concerning the hyperparameters of the PBT algorithm, they were set as defined below.:

- Perturbation interval: 4;
- Quantile fraction: 0.125;
- Resample probability: 0.5.

The number of PBT and RL workers used in these experiments were set to 16 and 2, respectively, unless stated otherwise. The implementation of PBT used in this work is available in the library *Ray* (LIANG et al., 2017). The aforementioned hyperparameter setup was defined after preliminary experiments and a hyperparameter analysis, which is presented and discussed later in this work. The three hyperparameters of the PBT alongside the number of PBT and RL workers form the set of five hyperparameters of our method that need to be set to an exact value (any hyperparameter of the underlying RL algorithm can be set to a flexible range to be controlled by PBT). In other words, they are the hyperparameters of the last and highest level of control.

5.1.3 Choosing the Metaheuristics and the Optimization Problems

The algorithm HCLPSO was used to solve all 29 functions of the CEC17 Bound Constrained Benchmark Continuous Functions in their versions with 10 dimensions (ALI et al., 2016). It is important to mention that all functions in this set are minimization problems. It means that the objective functions are multiplied by -1 to make it suitable for the proposed method. In this work, the instances are numbered according to the number of their corresponding functions in the competition.

In order to assess the generality of the proposed method, four other algorithms were used: FSS, DE with DE/rand/1/bin strategy, binary GA, and ACO. The FSS and DE algorithms were used to solve the same 29 functions of the aforementioned CEC17 benchmark set with 10 dimensions. Therefore, HCLPSO, FSS, and DE form the continuous part of our benchmark set, which has 87 instances to test our method (*i.e.* the number of combinations between 3 algorithms and 29 problems). Concerning the problems for the binary GA, 23 instances of the Knapsack problem have been used. These instances have 20 or 50 items and were randomly selected from (PISINGER, 2005), where the definition of the problem and the description of all instances can be found. The objective function of the Knapsack problem must be maximized. It is important to mention that, in our implementation, the fitness values of the unfeasbile solutions are set to zero. The ACO algorithm was used to solve 23 instances of the Traveling Salesman Problem (TSP) with 10, 30, and 50 cities. The distances between the cities were randomly generated. In (ILAVARASI; JOSEPH, 2014), it can be found a description of the TSP. Even though TSP is a minimization problem, its objective function is calculated as the inverse of the original fitness function, as defined in (DAS; MULLICK; SUGANTHAN, 2016). The definition of each of the selected instances is available at the following link: https://github.com/lacerdamarcelo/rl_based_parameter_control_ea_si /raw/main/instances_description.ods. The spreadsheet available in this link makes reference to the instance files that are available at the following repositories: TSP: https://github.com/ lacerdamarcelo/rl_based_parameter_control_ea_si/tree/main/vlsi_tsp/mar_tsp; Knapsack Problem: https://github.com/lacerdamarcelo/rl_based_parameter_control_ea_si/tree/ main/knapsack_problem.

It can be noticed that the number of dimensions of the problems used in this benchmark set is low. However, the objective of such a setup is to assess the capacity of our approach to work with scenarios with different algorithms and problems. In total, 133 scenarios have been chosen to evaluate our proposed method.

As previously mentioned, the values of the parameters defined for the metaheuristics by the static policies were computed through a state-of-the-art tuning method. The chosen method is a distributed implementation of the algorithm I/F-Race (BALAPRAKASH; BIRATTARI; STÜTZLE, 2007b). In our straightforward implementation, each parameter setup is evaluated by a single thread. Besides, the number of allocated threads was defined as the number of available CPUs in the system. Therefore, if 48 physical or virtual CPUs are allocated for the tuning task, only 48 configurations can be evaluated for execution of the F-Race part of the I/F-Race algorithm (*i.e.* they are all evaluated in parallel).

For our method and the I/F-Race algorithm, for each instance $f \in T_i$, where T_i is the i-th problem in the experimental benchmark (CEC17 continuous functions, Knapsack

Problem or TSP), the remaining instances $T - \{f\}$ are used as training instances. All training processes for both methods were executed during 24 hours. In the end of each training process of the I/F-Race algorithm, for each function of the benchmark, the last surviving setup is chosen. If more than one setup lasts, one of them is randomly chosen. The parameter setup for the I/F-Race algorithm is depicted below:

- Number of parameter configurations evaluated for each new F-Race process: 48;
- Minimum F-Race iterations before it starts removing bad setups: 20;
- Parameter setup generator standard deviation: 0.3 * range of values of the parameter;
- Minimum number of configurations in the F-Race pool: 10;
- Maximum number of F-Race iterations: 30.

For all approaches tested in this work, each policy, both dynamic and static, evaluated in a given metaheuristic and optimization problem, is made by executing it 30 times. For almost all cases, 300 iterations were set as budget to the metaheuristic, unless stated otherwise. Moreover, for every experiment, the population size was set to 100 candidate solutions. For the HCLPSO, the exploration and exploitation population sizes were set to 50 each. The population size was fixed even in the dynamic policies because the complexity added to the optimization process when the population size is dynamically adjusted, especially in the swarm-based algorithms, is not the focus of this work.

Regarding the human-designed policies, they were defined based on relevant papers and a few common sense values as previously mentioned. These values and the reference papers that were used to define the given policies are depicted below:

- HCLPSO (LYNN; SUGANTHAN, 2015):
  - w: Linear decrease from 0.99 to 0.2;
  - c: Linear decrease from 3 to 1.5;
  - candidate solution step: Linear decrease from 0.1 to 0.000001;
  - $c1$: Linear decrease from 2.5 to 0.5;
  - $c2$: Linear increase from 0.5 to 2.5;
  - m: 5.
- FSS (FILHO et al., 2009b):
  - candidate solution step: Linear decrease from 0.1 to 0.000001;
  - Volitive step: twice the candidate solution step;

  - Maximum weight: 5000.

- DE (DAS; MULLICK; SUGANTHAN, 2016):
  - F: 2;
  - Crossover probability: 0.5;
- ACO (DAS; MULLICK; SUGANTHAN, 2016):
  - α: 1;
  - β: 2;
  - ρ: 0.98;
  - Probability of using the best ant ever to update the pheromone trail instead of the best in the iteration: linear increase from 0 to 1.
- GA (MICHALEWICZ; ARABAS, 1994; HRISTAKEVA, 2004):
  - Mutation probability: 0.1;
  - Crossover probability: 0.75;
  - Elitism size: 2.

For the policies trained with our method and the random policies, the numerical ranges from which these policies must choose a value to be set to the parameters of the metaheuristics were defined as follows:

- HCLPSO:
  - w: [0.2, 0.99];
  - c: [1.5, 3];
  - $c1$: [0.5, 2.5];
  - $c2$: [0.5, 2.5];
  - m: 5.
- FSS:
  - candidate solution step: [0, 0.1];
  - Volitive step [-0.2, 0.2] (the RL algorithm decides whether to contract or not);
- DE:
  - F: [0.01, 4];
  - Crossover probability: [0.01, 1].

- ACO:
  - α: [0, 4];
  - β: [0, 4];
  - ρ: [0, 1];
  - Probability of using the best ant ever to update the pheromone trail instead of the best in the iteration: [0, 1].
- GA:
  - Mutation probability: [0.001, 1];
  - Crossover probability: [0.001, 1];
  - Elitism size: [1, 5].

All comparisons between policies in a given metaheuristic and a given optimization problem were performed through the Wilcoxon Rank-Sum test with 95% of confidence with a sample size of 30, unless stated otherwise.

It is important to mention that in the experiments carried out in this work, all functions have been used once as unseen function. When a function is used as an unseen problem, the remaining functions in the benchmark set are used as the training set. For example, when function 1 from the CEC17 benchmark set is used as testing function, the remaining functions 3-30 are used as training set.

5.1.4 The Computational Setup

The experiments were performed on the PALMA-II HPC cluster, from University of Muenster, Germany. At the time when the experiments have been made, the system had 412 nodes with 2 Intel Xeon Gold 6140 CPUs containing 18 physical CPU cores each, what allows up to 72 threads running in parallel per node. The connection between nodes was made through Intel Omni-Path with 100Gbit/s of bandwidth. The workload of the system was managed by Slurm 18.08.8. For each experiment, one CPU was allocated for each allocated thread. Thus, for our method and the I/F-Race algorithm, 48 CPUs were allocated. Finally, the following softwares, operating system, libraries and their corresponding versions were used: CentOS 7, Python 3.6.6, GCC 7.3.0, OpenMPI 3.1.1, Ray 0.8.4 (RLLib), Tensorflow 2.1.0, Numpy 1.18.1 and Scipy 1.4.1.

5.2 RESULTS AND DISCUSSIONS

The experiments presented in this study have been divided into three parts:

1. Hyperparameter analysis: The values of the hyperparameters of the PBT algorithm (*i.e.* the last layer of control) are varied in order to assess their effects on the performance of the policy learning process.

2. Analysis of different budgets in the training and the testing phases: In these experiments, policies trained with 100 iterations per episode are tested with 300 iterations. Such experiments are intended to verify if it is possible to save training time by reducing the budget per training episode without affecting the performance of the trained policies in an unseen function.

3. Generality assessment: The generality of our method is assessed by testing it on 133 different combinations of metaheuristics and optimization problems. Five continuous, binary and combinatorial metaheuristics are used to build the experimental benchmark. Moreover, our method is compared with static policies tuned by a state-of-the-art tuning algorithm, a random policy and a human-designed policy, which are defined by the authors of relevant papers in the literature. The comparisons against the state-of-the-art tuning method are intended to compare the proposed dynamic parameter adjustment method against a static policy tuned by a well-known and widely used algorithm. Comparing against the human-designed policies has the objective of comparing a computer-designed parameter adjustment policy against a human-designed one. Finally, comparing against a random policy aims at analyzing how far are the trained policies from a completely random and clueless policy.

5.2.1 Hyperparameter analysis

In this part of the experiment, only HCLPSO is used. As previously mentioned, HCLPSO is used to solve the already presented 29 CEC17 functions. In this section, we analyze the effects on the quality of the trained policies and, consequently, on the performance of the HCLPSO algorithm, when each one of the five hyperparameters of the proposed method are varied.

5.2.1.1 Number of PBT workers

The first hyperparameter analyzed in this section is the number of PBT workers. For such experiments, the training process was executed with 4, 8 and 16 PBT workers. The remaining hyperparameters were fixed as defined in the experimental methodology description. The performance of the trained policies for each problem is defined as the average best fitnesses found by the HCLPSO algorithm for the problem at hand across 30 executions. The performances of the three setups were compared to each other.

Table 6 shows the number of comparisons between the $p * 100\%$ best policies trained with each tested number of PBT workers where the setup defined by the column name significantly outperformed another one. If A significantly outperforms B, it means that not only the performance of A was superior to B, but also the p-value computed with Mann-Whitney U test is lower than or equal to 0.05. Such a test was used in the experiments with all quantiles except *Best*, which used Wilcoxon Rank-Sum test with 5% of confidence. The *best* results are achieved by the policy among the pool of trained policies that showed the highest performance on the testing function. Given that this function is unknown in a real world situation, such a especial policy selection method cannot be used in real cases. This is used in this work only to show how good the policies generated by our method can be, and how well the underlying metaheuristic can perform if an excellent policy selection method is used.

Since each setup is compared with the other two setups in 29 functions, 58 comparisons are made for each population size of PBT workers. It means that, for instance, when the top 30% best policies trained with 4, 8, and 16 PBT workers, were compared to each other, the performance of the policies trained with 4 PBT workers significantly outperformed one of the other setups in 43 out of 58 comparisons. Besides, comparing the same superior quantile of trained policies, the setup with 8 PBT workers outperformed another setup in 16 cases, while the setup with 16 PBT workers showed significant superiority in 24 comparisons. It is important to mention that comparing the 30% best policies means that the best fitness found in each of the 30 executions performed in the testing function by each policy among the 30% best policies is included in the samples to be compared.

Table 6 shows the number of comparisons between the $p * 100\%$ best policies for each tested perturbation interval, where they outperformed another setup and scored 0.05 or less of p-value computed through Mann-Whitney U test (all quantiles from 1.0 to 0.01) or Wilcoxon Rank-Sum test (only comparisons with the best policy). It is important to mention that, in order to make such comparisons, the available trained policies are ranked by their performances in the testing function itself. Therefore, to compare the 30% best policies in function 1, the available trained policies (*i.e.* the policies trained with the remaining benchmark functions) are tested in the function 1 itself, and their performances are used to define the 30% best policies. Surely, since in the real world the testing function is unknown, this is made only for experimental purposes.

It can be seen in Table 6 that up to the comparison of the 10% best policies, the setup with 4 PBT workers performed better than the other setups in more cases. However, when only the top policies are compared, the setup with 16 PBT workers outperformed the other configurations in more comparisons. Such a superiority of smaller populations of PBT workers is surprising, since the conclusions drawn in the paper available in the Appendix B of this work clearly state that the more PBT workers, the better. Nevertheless, it is clear that the more restrictive the group of top policies, the less sensitive is the HCLPSO to the number of PBT workers.

Table 6 – Number of comparisons between the $p * 100\%$ best policies for each tested perturbation interval, where they outperformed another setup and scored 0.05 or less of p-value computed through Mann-Whitney U test (all quantiles from 1.0 to 0.01) or Wilcoxon Rank-Sum test (only comparisons with the best policy).

	Number PBT workers		
Quantile (p)	**4**	**8**	**16**
1.0	**34**	33	15
0.9	**37**	32	12
0.8	**39**	27	14
0.7	**41**	26	14
0.6	**42**	24	16
0.5	**45**	21	19
0.4	**45**	19	22
0.3	**43**	16	24
0.2	**39**	15	24
0.1	**38**	18	23
0.01	22	10	**25**
Best	3	5	**11**

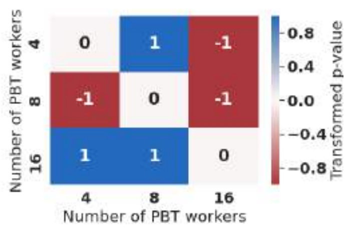
Source: Produced by the author.

Figures 32 and 33 in Appendix A.1 show the comparisons between the three tested sizes of the population of PBT workers for functions 1-16 and 17-30, respectively. Such comparisons are made with the 30% best policies in the pool of trained policies. However, the values shown in the figures are not the original p-values of the hypothesis tests. A transformed p-value is calculated for each comparison in order to make the results visually more intuitive. Let A be the proposed method with the number of PBT workers defined in the row's name of the heatmaps, and let B be the proposed method with the population size of PBT workers defined in the column's name. If A outperforms B, the transformed p-value is $1 - p$, where p is the original p-value. Otherwise, the transformed p-value is calculated as $p - 1$. The closer to 1 the transformed p-value, the higher the probability of A being superior to B. On the other hand, the closer to -1, the higher the probability of B being superior to A. The closer to zero, the closer to each other the performances of A and B. Figures 34 and 35 in Appendix A.1 show the transformed p-values of the comparisons between the best policies trained on each setup. It is clear in such figures that the performance of the best policies is less sensitive to the number of PBT workers than the performance of the 30% best policies, as already observed in Table 6. It means that the better the policy selection method, *i.e.* the higher the probability of choosing the top policies in a pool of trained policies, the less sensitive to the number of PBT workers the final performance of the optimization process. Figure 13 delivers a sample from Figures 32 and 33, while Figure 14 shows some of the results presented in Figures 34 and 35. Tables 12 and 13 shows the numerical results of these experiments for further analysis.

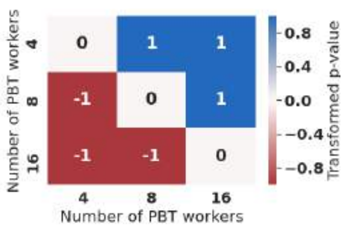
As previously presented, comparing up to the 10% best policies, the policies trained with 4 PBT workers performed better than the policies trained with 8 and 16 PBT

Figure 13 – Sample from Figures 32 and 33 in Appendix A.1. Comparing the 30% best policies trained with different sizes of the population of PBT workers: 4, 8, and 16. The comparisons have been made between samples of the best fitness found in the executions of the HCLPSO algorithm with the 30% best trained policies of the training set available for each function. The original p-values were computed with the Mann-Whitney U test.

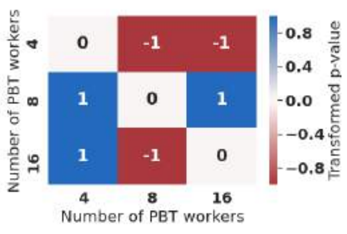
(a) Function 1



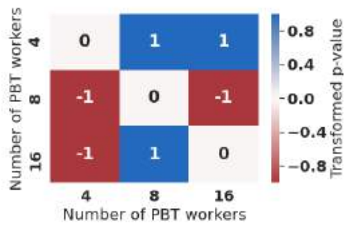
(b) Function 3



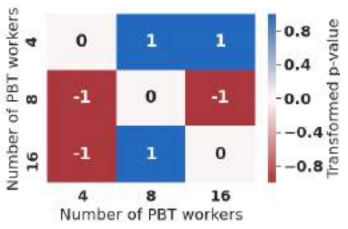
(c) Function 4



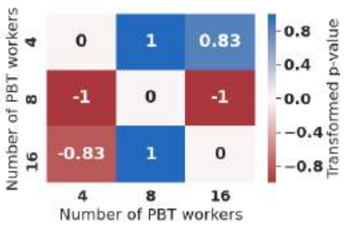
(d) Function 5



(e) Function 6



(f) Function 7

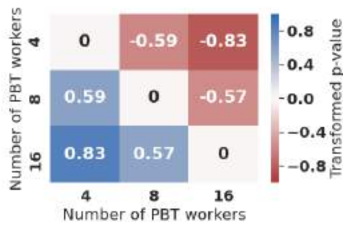


Source: Produced by the author.

workers in most of the cases. This difference is probably due to the fact that more PBT workers cause the policy search to explore the policy space more widely. Performing a wider search creates policies with more diversified performance, which means that not only better policies are more likely to be found, but also worse policies. Figure 15 shows three examples of such a behavior in functions 9, 14, and 27. These boxplots show the training rewards accumulated during the training process of all policies in the policy pool available for functions 9, 14, and 27. It can be seen that the maximum training reward (*i.e.* the quality of the best policy) is consistently higher with 16 PBT workers than with 4 PBT workers. However, the distribution of training reward is more spread out across the y axis when the population is large. The same behavior was observed in all other cases. Table 7 shows for all functions the maximum reward and the standard deviation of the reward distribution. In this table, the aforementioned behavior can also be noticed: greater maximum reward and standard deviation with 16 PBT workers than with 4 PBT workers. Another explanation for such results is the fact that, in the implementation used in this work, the number of PBT workers defines the frequency in which the target network is updated. It is possible that the configuration with 4 PBT workers may have “accidentally" set such a hyperparameter to a good value, what compensates the lack of diversity in the policy search.

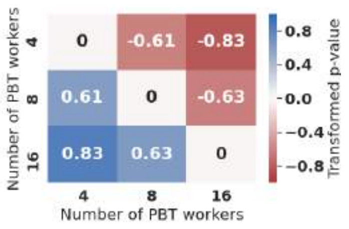
Figure 14 – Sample from Figures 34 and 35 in Appendix A.1. Comparing the best policies trained with different sizes of the population of PBT workers: 4, 8, and 16. The comparisons have been made between samples of the best fitness found in the executions of the HCLPSO algorithm with the 30% best trained policies of the training set available for each function. The original p-values were computed with the Mann-Whitney U test.

(a) Function 8



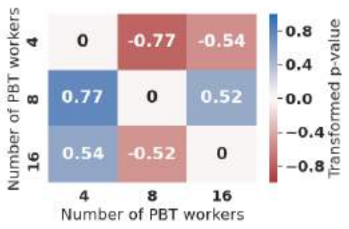
	4	8	16
4	0	-0.59	-0.83
8	0.59	0	-0.57
16	0.83	0.57	0

(b) Function 10



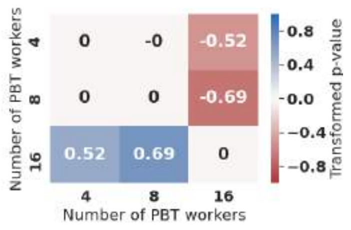
	4	8	16
4	0	-0.61	-0.83
8	0.61	0	-0.63
16	0.83	0.63	0

(c) Function 11



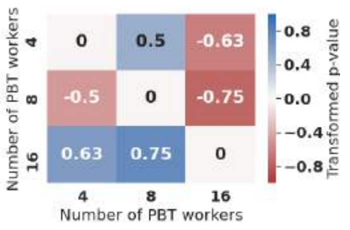
	4	8	16
4	0	-0.77	-0.54
8	0.77	0	0.52
16	0.54	-0.52	0

(d) Function 12



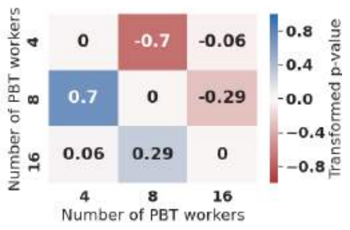
	4	8	16
4	0	-0	-0.52
8	0	0	-0.69
16	0.52	0.69	0

(e) Function 14



	4	8	16
4	0	0.5	-0.63
8	-0.5	0	-0.75
16	0.63	0.75	0

(f) Function 15

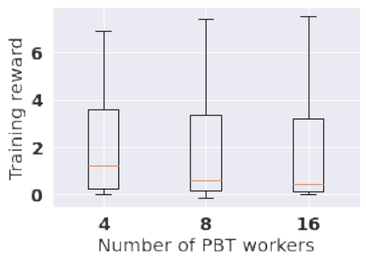


	4	8	16
4	0	-0.7	-0.06
8	0.7	0	-0.29
16	0.06	0.29	0

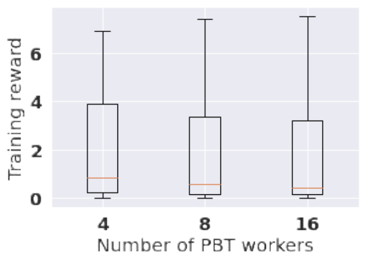
Source: Produced by the author.

Figure 15 – Training rewards accumulated during the training process of each policy.

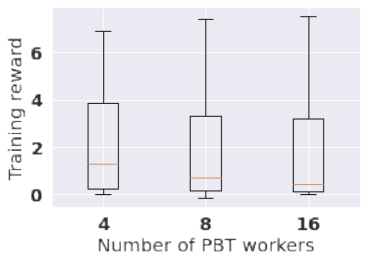
(a) Function 9



(b) Function 14



(c) Function 27



Source: Produced by the author.

Table 7 – Maximum training reward and standard deviation of the accumulated training rewards of all trained policies available for each function.

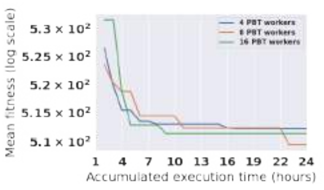
	4 PBT workers		**8 PBT workers**		**16 PBT workers**	
Function	**Standard Deviation**	**Maximum Reward**	**Standard Deviation**	**Maximum Reward**	**Standard Deviation**	**Maximum Reward**
1	2.15	6.91	2.20	7.29	2.13	7.49
3	1.91	6.22	2.05	7.39	2.13	7.54
4	2.27	6.91	2.29	7.39	2.33	7.54
5	2.18	6.91	2.27	7.39	2.31	7.54
6	2.19	6.91	2.26	7.39	2.31	7.54
7	2.19	6.91	2.27	7.39	2.31	7.54
8	2.19	6.91	2.26	7.39	2.31	7.54
9	2.19	6.91	2.28	7.39	2.32	7.54
10	2.18	6.91	2.27	7.39	2.31	7.54
11	2.20	6.91	2.28	7.39	2.33	7.54
12	2.14	6.91	2.22	7.39	2.28	7.54
13	2.12	6.91	2.15	7.39	2.20	7.54
14	2.23	6.91	2.29	7.39	2.33	7.54
15	2.18	6.91	2.25	7.39	2.29	7.54
16	2.20	6.91	2.27	7.39	2.31	7.54
17	2.17	6.91	2.27	7.39	2.31	7.54
18	2.10	6.91	2.14	7.39	2.18	7.54
19	2.10	6.91	2.17	7.39	2.21	7.54
20	2.18	6.91	2.26	7.39	2.30	7.54
21	2.17	6.91	2.26	7.39	2.30	7.54
22	2.18	6.91	2.27	7.39	2.31	7.54
23	2.18	6.91	2.27	7.39	2.31	7.54
24	2.18	6.91	2.26	7.39	2.31	7.54
25	2.19	6.91	2.28	7.39	2.32	7.54
26	2.19	6.91	2.27	7.39	2.31	7.54
27	2.20	6.91	2.27	7.39	2.31	7.54
28	2.17	6.91	2.26	7.39	2.30	7.54
29	2.21	6.91	2.29	7.39	2.32	7.54
30	2.22	6.91	2.28	7.39	2.31	7.54

Source: Produced by the author.

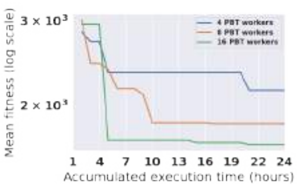
Figures 36 and 37 show the mean best fitness found by HCLPSO solving each CEC17 benchmark function controlled by the best policy found with 4, 8, and 16 PBT workers *until* different points in time. The average best fitnesses are calculated from 30 executions of the metaheuristic with the best policy so far in each of the tested functions. In 16 out of 29 functions, the performances of the best policies found by each setup are very close to each other during almost the entire training process. Figure 16.a exemplifies such a pattern with function 5. In 8 problem instances, the configuration with 16 PBT workers outperforms the other two configurations during most of the training time. It means that if an early stopping mechanism is implemented, the configuration with 16 PBT workers is more likely to return a better best policy. Finally, in 2 problem instances, the setup with 4 PBT workers outperformed the other two setups. Figures 16.b and 16.c illustrate these groups with functions 13 and 29, respectively.

Figure 16 – Sample from Figures 36 and 37 in Appendix A.1. Mean fitness of the best policy found after some time of training in hours with 4, 8, and 16 PBT workers. Each plotted point in the lines shows the mean best fitness found by HCLPSO controlled by the best policy found so far.

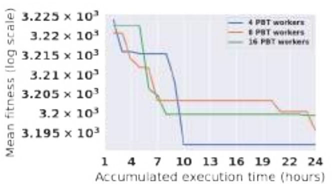
(a) Function 5



(b) Function 13



(c) Function 29



Source: Produced by the author.

5.2.1.2 Number of RL workers

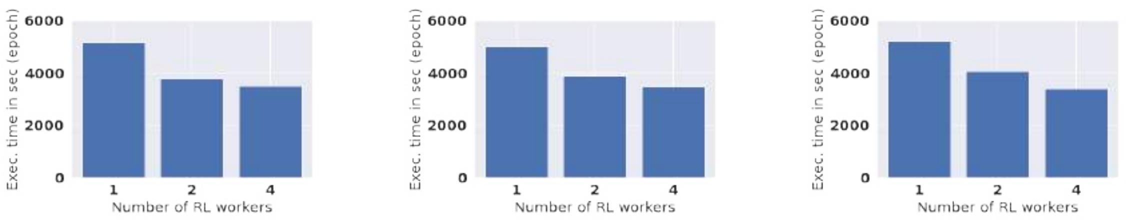
As already mentioned in this work, the number of RL workers allocated for each PBT worker affects the time efficiency of the training process. With more RL workers, the experience collection and, therefore, the training epoch becomes shorter. It means that the decision regarding the adjustment of such a hyperparameter is straightforward: the more, the better. Surely, since the number of workers that can be executed in parallel in any computing platform is limited, there is a trade-off in the cases where it is recommended to use a large population of PBT workers. The more PBT workers are used, *i.e.* the more diversified is the policy search, the less RL workers per PBT worker can be allocated, *i.e.* the less time-efficient is the training process. The more RL workers are used, the faster the training process runs, but the less diverse the policy search.

Figures 38 and 39 in Appendix A.2 show the average execution time of one training epoch with 1, 2, and 4 RL workers. The corresponding function of each bar chart is the function that was used to train the policies. The figures clearly show that there is always a speed-up when the number of RL workers is increased. Figure 17 shows three of the 29 functions shown in Figures 38 and 39, which show slightly different speed-ups. It is important to mention that the testing phase took on average approximately 3000 seconds to be executed, regardless of the number of RL workers. In the experiments carried out in this study, the sets of training functions used to test the training policies during the learning process for each testing functions are similar to each other. Besides, the fitness call of the benchmark functions are quite similar to each other in terms of execution time. Thus, there was no significant variance on the execution time of the testing phase across the training process of each benchmark function. Therefore, it is clear that the testing phase is a bottleneck for the training process as a whole, since it takes a large portion of the execution time of a single training iteration. Without such a bottleneck, speed-ups approximately proportional to the number of RL workers could be achieved.

Despite the aforementioned nonproportionality, it is clear that the proposed architecture benefits from parallel computing platforms. Therefore, it satisfactorily addresses the issue number 1 found in the literature. This is expected to happen since this architecture is based on well-succeeded distributed implementations of RL algorithms. However, a more efficient implementation should be made in order to achieve greater speed-ups.

Figure 17 – Sample from Figures 38 and 39 in Appendix A.2. Average execution time in seconds of one training epoch during the training process with 1, 2, and 4 RL workers.

(a) Function 1 (b) Function 15 (c) Function 29



Source: Produced by the author.

5.2.1.3 Perturbation interval

As previously mentioned in this work, the perturbation interval is the hyperparameter of the PBT algorithm that defines how often the PBT workers will communicate with each other. The lower its value, the more often such a communication occurs. In the communication process, not only parameters and hyperparameters are exchanged between workers, what causes the algorithm to exploit promising regions, but the hyperparameters are also perturbed according to a given probabilistic distribution. It is clear that such an operator is a hybrid mechanism regarding the trade-off between exploration and exploitation. Therefore, it is not clear how to deal with such a balance by adjusting the perturbation interval.

Table 8 shows the number of comparisons between the $p * 100\%$ best policies for each tested perturbation interval, where they outperformed another setup and scored 0.05 or less on the p-value computed through Mann-Whitney U test (all quantiles from 1.0 to 0.01) or Wilcoxon Rank-Sum test (only comparisons with the best policy). It can be seen that the closer to the top the selected policies, the better the policies trained with 8 iterations of perturbation interval in comparison to the others. It can be seen that the recommended perturbation interval gradually increases from 2 to 8 as the quantile decreases. Moreover, the lower the selected quantile, the smaller the difference between the performances of the tested perturbation intervals.

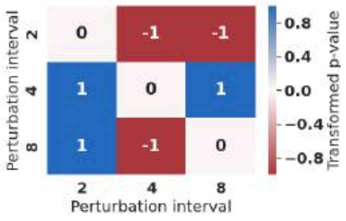
Figures 40 and 41 in Appendix A.3 show the transformed p-values of the 30% best policies trained with perturbation intervals of 2, 4, and 8 training iterations, tested in functions 1-16 and 17-30, respectively. The original p-values were calculated with the Mann-Whitney U test. Figures 42 and 43 in Appendix A.3 show the transformed p-values

Table 8 – Number of comparisons between the $p * 100\%$ best policies for each tested perturbation interval, where they outperformed another seutp and scored 0.05 or less of p-value computed through Mann-Whitney U test (all quantiles from 1.0 to 0.01) or Wilcoxon Rank-Sum test (only comparisons with the best policy).

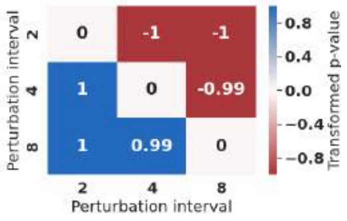
	Perturbation interval		
Quantile (p)	**2**	**4**	**8**
1.0	**49**	34	1
0.9	**45**	36	2
0.8	37	**39**	5
0.7	36	**41**	4
0.6	29	**43**	10
0.5	25	**44**	12
0.4	28	**44**	12
0.3	19	**41**	19
0.2	17	**35**	20
0.1	16	**26**	24
0.01	15	16	**20**
Best	2	1	**3**

Source: Produced by the author.

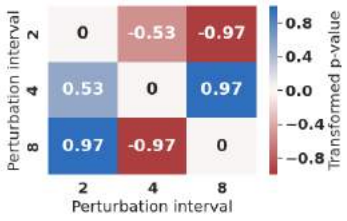
of HCLPSO executed with the best policy trained with perturbation intervals of 2, 4, and 8 iterations. Figures 18 and 19 show some of the comparisons presented in Figures 40 and 41, and 42 and 43, respectively. It is clear that the lower the quantile of the selected policies, the less sensitive the HCLPSO to the perturbation interval. Therefore, it can be concluded that the better the policy selection method, *i.e.* the higher the probability of choosing the top policies in a pool of trained policies, the less sensitive to the hyperparameter at hand the final performance of the optimization algorithm. Tables 14 and 15 deliver the numerical results of such experiments for further analysis.



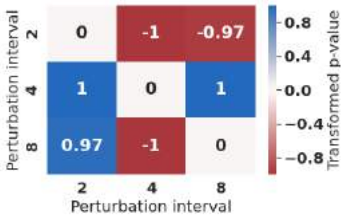
(a) Function 11



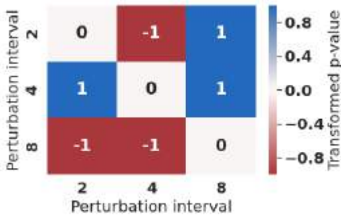
(b) Function 12



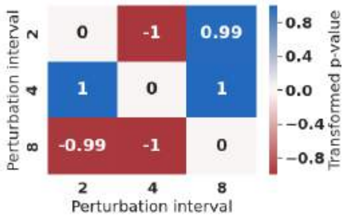
(c) Function 13



(d) Function 14



(e) Function 15



(f) Function 16

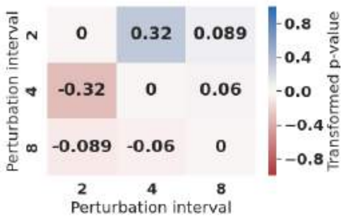
Figure 18 – Sample from Figures 40 and 41 in Appendix A.3. Comparing the 30% best policies trained with different perturbation intervals: 2, 4, and 8 iterations. The comparisons have been made between samples of the best fitness found in the executions of the HCLPSO algorithm. The original p-values were computed with the Mann-Whiteney U test.

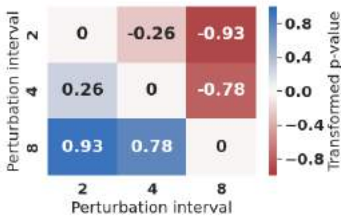
Figure 19 – Sample from Figures 42 and 43 in Appendix A.3. Comparing the best policy trained with different perturbation intervals: 2, 4, and 8 iterations. The comparisons have been made between samples of the best fitness found in the executions of the HCLPSO algorithm solving functions 1-16 from the CEC17 benchmark set. The original p-values were computed with the Wilcoxon Rank-Sum test.

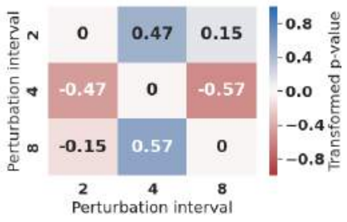
(a) Function 1

(b) Function 3

(c) Function 4



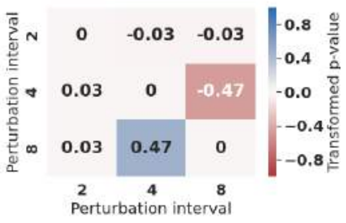


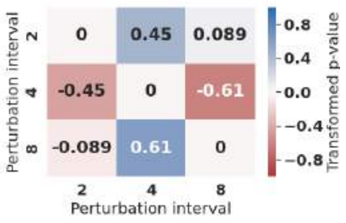


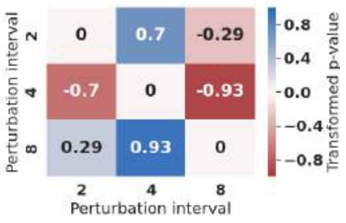
(d) Function 5

(e) Function 6

(f) Function 7





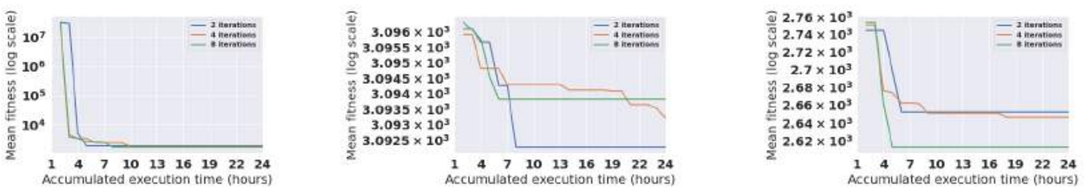


Source: Produced by the author.

Figures 44 and 45 in Appendix A.3 show the mean best fitness found by HCLPSO solving each CEC17 benchmark function controlled by the best policies found with 2, 4, and 8 iterations of perturbation interval at different points in time. The mean best fitness is calculated from 30 executions. The performance of the three tested configurations remained similar to each other during most of the training process in 17 functions, as exemplified in Figure 20.a with function 1. The configuration with 2 iterations presented a significantly superior best policy during most of the training process in only 2 functions, as shown in Figure 20.b with function 27. Even though the training process with 4 iterations generated the 30% best policies among all tested configurations, it could not keep its best policy superior to the best policies found by the other configurations for most of the training time in any of the benchmark problems. Regarding the setup with 8 iterations, it overcame the other configurations during most of the 24 hours of training in only 3 functions, as illustrated in Figure 20.c with function 24. After analyzing such results, it can be concluded that the adjustment of the perturbation interval does not cause a great impact on the evolution of the best policy found for the majority of the functions. It means that the quality of the best policy found in a training process with early stopping is not seriously affected by such a hyperparameter.

Figure 20 – Sample from Figures 44 and 45 in Appendix A.3. Mean fitness of the best policy found after some time of training in hours with 2, 4, and 8 iterations of perturbation interval. Each plotted point in the lines shows the mean best fitness found by HCLPSO controlled by the best policy found so far.

(a) Function 1 (b) Function 27 (c) Function 24



Source: Produced by the author.

5.2.1.4 Quantile fraction

Let q be the value set as the quantile fraction for a given training process. In the exploitation phase, the $q\%$ worst PBT workers copy the hyperparameters and parameters of the $q\%$ best PBT workers. It means that the greater the quantile fraction, the more PBT workers exchange information and, therefore, the stronger the exploitation.

Table 9 provides the number of comparisons between the $p{*}100\%$ best policies for each tested quantile fraction, *i.e.* 0.125, 0.25, and 0.375, where it outperformed other setup and the p-value computed through Mann-Whitney U test (for $0.01 \leq p \leq 1.0$, where p is the quantile of the selected policies) or Wilcoxon Rank-Sum test (for the comparisons between the best policies) is lower than or equal to 0.05. It is straightforward to see the the lower the quantile fraction, the better. It means that the less PBT workers communicate with each other, the more diverse is the search process and, thus, the higher the probability of finding a good policy. Surely, that must have a minimum quantile fraction recommended, otherwise the behavior of PBT would be very close to the previously presented Random/-Grid Search. Such a value should be found through a more thorough parameter analysis, *i.e.* with a higher granularity on the discretization of the hyperparameter space. It is important to highlight that the more restrictive the set of top selected policies, the less sensitive to the quantile fraction the performance of HCLPSO.

Table 9 – Number of comparisons between the $p * 100\%$ best policies for each tested quantile fraction, *i.e.* 0.125, 0.25, and 0.375, where it outperformed other setup and the p-value computed through Mann-Whitney U test (for $0.01 \leq p \leq 1.0$, where p is the quantile of the selected policies) or Wilcoxon Rank-Sum test (for the comparisons between the best policies) is lower than or equal to 0.05.

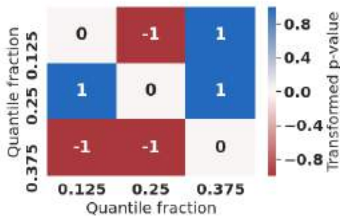
	Quantile fraction		
Quantile (p	**0.125**	**0.25**	**0.375**
1.0	**46**	21	16
0.9	**46**	18	20
0.8	**47**	18	17
0.7	**49**	20	15
0.6	**50**	20	15
0.5	**51**	20	15
0.4	**52**	20	12
0.3	**48**	22	11
0.2	**48**	26	7
0.1	**48**	26	6
0.01	**27**	19	7
Best	**10**	3	1

Source: Produced by the author.

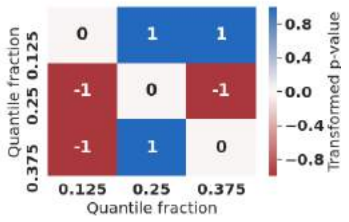
Figures 46 and 47 in Appendix A.4 show the transformed p-values achieved by the HCLPSO controlled with the 30% best policies, trained with three quantile fractions, 0.125, 0.25, and 0.375. The original p-values calculated for such comparisons have been computed through Mann-Whitney U test. Some of the comparisons shown in such figures are presented in Figures 46. Figures 48 and 49 show the transformed p-values of the comparisons made between the best policy found for each of the three already mentioned tested values for the quantile fraction. Some of these comparisons are shown in Figure 46. It is clear the superiority of the setup with 0.125 of quantile fraction and the low sensitivity of the top policies to such a hyperparameter. It means that the better the policy selection method, the less sensitive the performance of the chosen policy to the quantile fraction. Tables 16 and 17 provide the numerical results of these experiments for further analysis.

Figure 21 – Sample from Figures 46 and 47 in Appendix A.4. Comparing the 30% best policies trained with different quantile fractions: 0.125, 0.25, 0.375. The comparisons have been made between samples of the best fitness found in the executions of the HCLPSO algorithm. The original p-values were computed with the Mann-Whitney U test.

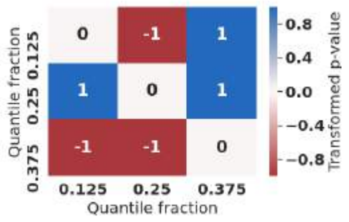
(a) Function 1



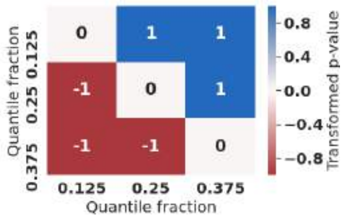
(b) Function 3



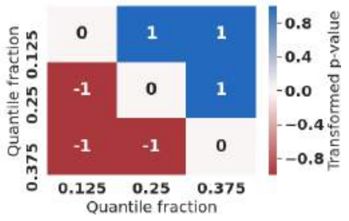
(c) Function 4



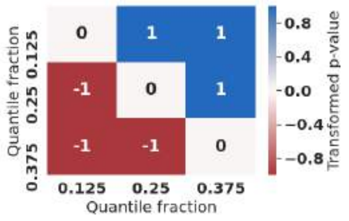
(d) Function 5



(e) Function 6



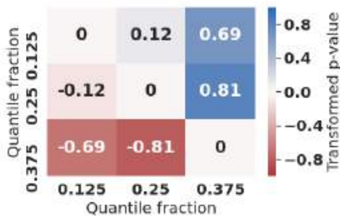
(f) Function 7



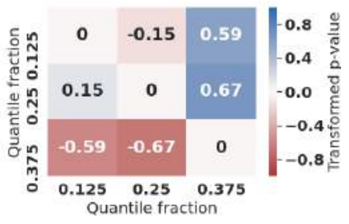
Source: Produced by the author.

Figure 22 – Sample from Figures 48 and 49 in Appendix A.4. Comparing the best policies trained with different quantile fractions: 0.125, 0.25, 0.375. The comparisons have been made between samples of the best fitness found in the executions of the HCLPSO algorithm. The original p-values were computed with the Mann-Whitney U test.

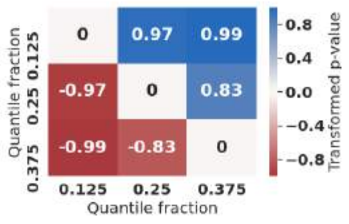
(a) Function 8



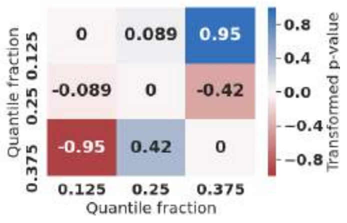
(b) Function 11



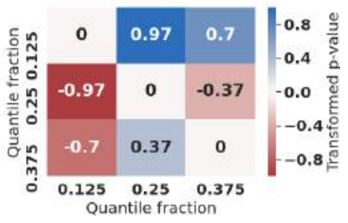
(c) Function 13



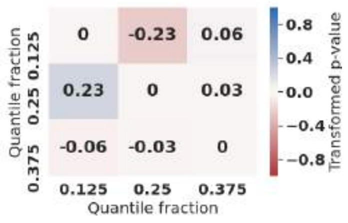
(d) Function 28



(e) Function 29



(f) Function 30

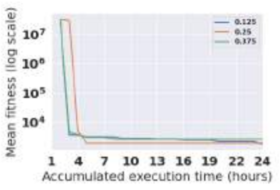


Source: Produced by the author.

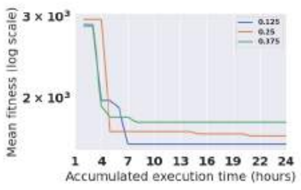
Figures 50 and 51 in Appendix A.4 show the mean best fitness found by HCLPSO solving each CEC17 benchmark function controlled by the best policy found with 0.125, 0.25, and 0.375 of quantile fraction at different points in time. The mean best fitnesses are calculated across 30 executions of the metaheuristic with the best policy so far in each of the tested functions. It can be seen that the best policy of each setup behaves similarly during the entire training process in 15 problem instances. In 6 functions, the training process with 0.125 of quantile fraction held the best policy among the best policies found by all configurations during most of the training process. The best policies found with 0.25 and 0.375 of quantile fraction remained superior to the others during most of the training process in only 3 instances each. It means that if early stopping is used, a training process with 0.125 of quantile fraction is slightly more likely to return a better policy. However, in most of the cases, the probability of finding the best policy among the three tested configurations for each setup would be very similar to each other.

Figure 23 – Sample from Figures 50 and 51 in Appendix A.4. Mean fitness of the best policy found after some time of training in hours with with quantile fractions of 0.125, 0.25, and 0.375. Each plotted point in the lines shows the mean best fitness found by HCLPSO controlled by the best policy found so far.

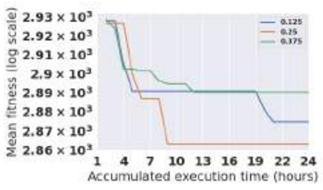
(a) Function 1



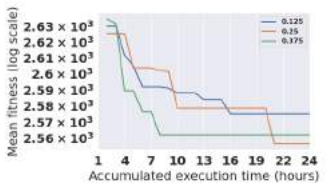
(b) Function 13



(c) Function 25



(d) Function 23



Source: Produced by the author.

5.2.1.5 Resample probability

The resample probability is the probability with which a set of hyperparameters is randomly perturbed according to their initial probability distribution in the exploration phase. As previously mentioned, the greater such a probability, the more explorative the policy search.

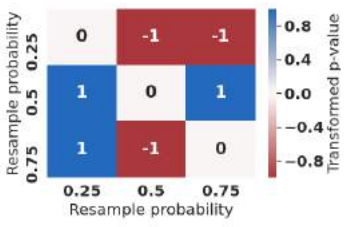
Table 10 provides the number of comparisons between the $p * 100\%$ best policies where each tested resample probability, *i.e.* 0.25, 0.5, and 0.75, outperformed other setup and the p-value computed through Mann-Whitney U test (for $0.01 \leq p \leq 1.0$, where p is the quantile of the selected policies) or Wilcoxon Rank-Sum test (for the comparisons between the best policies) is lower than or equal to 0.05. It can be seen that the performance of the setups with 0.5 and 0.75 are very similar to each other, although the policies trained with 0.5 of resample probability have shown better results in most of the selected quantiles. It can also be observed that the more restrictive the quantile, the higher the recommended resample probability, *i.e.* the more diverse must be the policy search, except when only the best policy is considered.

Table 10 – Number of comparisons between the $p * 100\%$ best policies where each tested resample probability, *i.e.* 0.25, 0.5, and 0.75, outperformed other setup and the p-value computed through Mann-Whitney U test (for $0.01 \leq p \leq 1.0$, where p is the quantile of the selected policies) or Wilcoxon Rank-Sum test (for the comparisons between the best policies) is lower than or equal to 0.05.

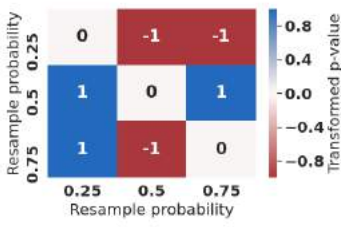
	Resample probability		
Quantile (p)	**0.25**	**0.5**	**0.75**
1.0	4	**42**	37
0.9	7	**42**	32
0.8	8	**43**	32
0.7	6	**43**	32
0.6	5	**45**	32
0.5	5	**44**	30
0.4	6	**42**	33
0.3	8	**36**	34
0.2	9	32	**40**
0.1	9	24	**42**
0.01	15	25	**28**
Best	**4**	**4**	1

Source: Produced by the author.

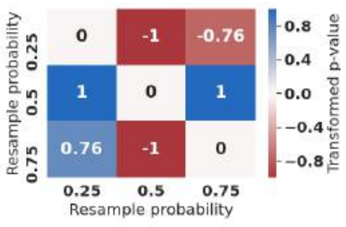
Figures 52 and 53 in Appendix A.5 show the transformed p-values computed for the comparisons between the 30% best policies trained with 0.25, 0.5, and 0.75 of resample probability. Figure 24 presents some of the comparisons provided in such figures. Figures 54 and 55 in Appendix A.5 show the transformed p-values computed for the comparisons between the best policy found in the training process executed with 0.25, 0.5, and 0.75 of resample probability, exemplified in Figure 25. Once again, the quality of the best policies shows a very low sensitivity to the hyperparameter at hand. However, it is also important to observe that despite the slight superiority of the setups with 0.25 and 0.5 of resample probability, when only p-values of 0.05 or less are considered, such setups presented the lowest number of comparisons with a positive transformed p-value. The policies trained with 0.25 of resample probability scored a positive transformed p-value in 18 of the 58 comparisons, while the setups with 0.5 and 0.75 achieved such a performance in 29 and 36 cases, respectively. This fact corroborates with the results observed in the analysis of the quantile fraction, which recommends high diversity in the policy search. Tables 18 and 19 in Appendix A.5 provide the numerical results of these experiments for further analysis.



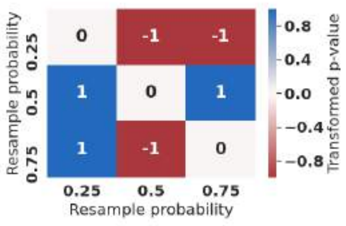
(a) Function 5



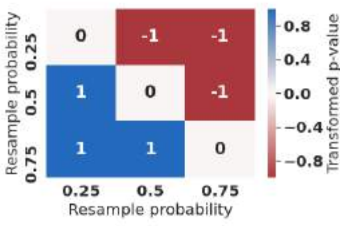
(b) Function 6



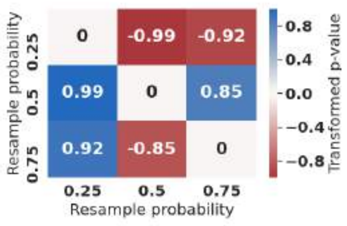
(c) Function 7



(d) Function 8



(e) Function 9

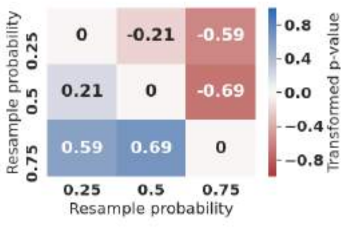


(f) Function 10

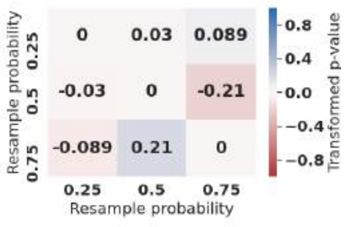
Figure 24 – Sample from Figures 52 and 53 in Appendix A.5. Comparing the 30% best policies trained with different resample probabilities: 0.25, 0.5, 0.75. The comparisons have been made between samples of the best fitness found in the executions of the HCLPSO algorithm. The original p-values were computed with the Mann-Whitney U test.

Figure 25 – Sample from Figures 54 and 55 in Appendix A.5. Comparing the best policy trained with different resample probabilities: 0.25, 0.5, and 0.75. The comparisons have been made between samples of the best fitness found in the executions of the HCLPSO algorithm. The original p-values were computed with the Wilcoxon Rank-Sum test.

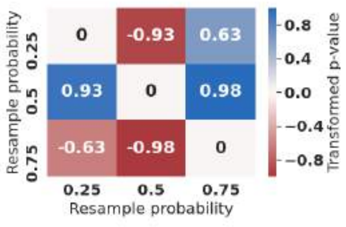
(a) Function 11



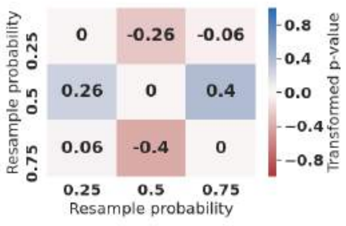
(b) Function 12



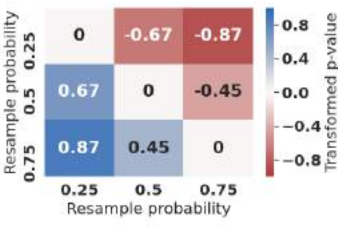
(c) Function 13



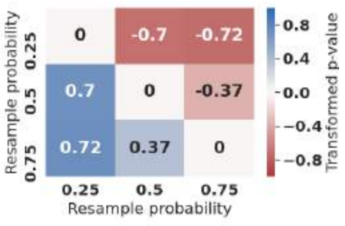
(d) Function 14



(e) Function 15



(f) Function 16



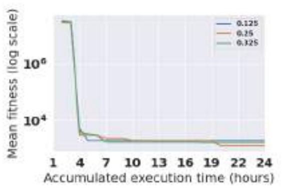
Source: Produced by the author.

Figures 57 and 58 in Appendix A.5 show the mean best fitness found by HCLPSO solving each CEC17 benchmark function controlled by the best policy found with 0.25, 0.5, and 0.75 of resample probability at different points in time. The mean best fitnesses are calculated across 30 executions of the metaheuristic with the best policy so far in each of the tested functions. Figure 26 shows four examples of the typical patterns found in these experiments.

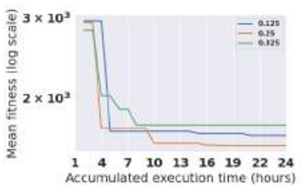
The best policy of each setup behaves similarly during the entire training process in 17 of the 29 functions, as illustrated in Figure 26.a with function 1. In 7 instances, the best policy found in the training process executed with 0.5 of resample probability overcame the best policies of the other configurations in most of the training time, as exemplified in Figure 26.b with function 13. The setups with 0.75 and 0.25 of resample probability made it in only 2 and 3 functions, respectively. Figures 26.c and 26.d show the experimental results for some of the functions of these groups. Therefore, if early stopping is used, there is a higher probability of finding the best policy among all best policies if the resample probability is set to 0.5. However, like in the previous experiments and comparisons made in this work, the hyperparameter at hand did not significantly affect the evolution of the quality of the best policy over time for the majority of the benchmark functions.

Figure 26 – Sample from Figures 57 and 58 in Appendix A.5. Mean fitness of the best policy found after some time of training in hours with 0.25, 0.5, and 0.75 of resample probability. Each plotted point in the lines shows the mean best fitness found by HCLPSO controlled by the best policy found so far.

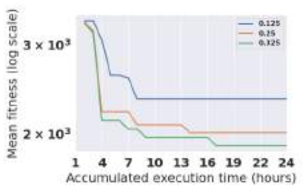
(a) Function 1



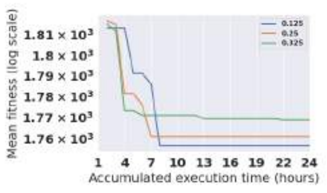
(b) Function 13



(c) Function 15



(d) Function 17



Source: Produced by the author.

5.2.1.6 Summarizing the hyperparameter analysis

A few important findings from the previously presented experiments are listed below:

- A large population of PBT workers does not always benefit the policy search. If the policy selection method does not often choose one of the top policies in the pool of trained policies, the performance of the optimization method in the unseen problem may be worse than if a small population had been chosen. For the policy selection methods that are more likely to choose one of the top policies, larger populations of PBT workers should be preferred. This is due to the fact that, in such cases, there is a higher probability of eventually finding a promising region in the policy space due to its greater exploration power. Nevertheless, the superiority of the policies trained with less PBT workers in a less restrictive selection (*i.e.* lower selection quantile) in comparison with the ones trained with larger populations is quite surprising. A deeper investigation should be carried out to better understand such a finding.

- More RL workers always speed up the training process, as long as the underlying hardware supports it. It means that the proposed architecture scales well and benefits from parallel computing platforms. Such a finding satisfactorily addresses the issue number 1 provided on the Introduction chapter of this work: training EA and SI parameter control policies with RL can be very computational-demanding. However, a more efficient implementation of the proposed method is needed to achieve more sizeable speed-ups.

- The remaining hyperparameters, *i.e.* perturbation interval, quantile fraction, and resample probability, should be adjusted aiming at the increase of exploration of the policy search. Therefore, low values should be set to the quantile fraction (*i.e.* 0.125, according to the previously discussed results). while the resample probability should be greater or equal than 0.5, according to the experiments. As already mentioned, it is hard to define the best value for the perturbation interval aiming at the increase of exploration of the search process, since the perturbation of the RL hyperparameters is always preceded by the exploitation phase. However, the intermediate value among the tested ones (*i.e.* 4) achieved the best performance in our experimental benchmark.

- For most of the tested functions, the performances of the top generated policies are not significantly affected by the adjustment of any of the hyperparameters of our method. It means that the better the policy selection method, the less sensitive to the adjustment of the hyperparameters the quality of the selected policy. Therefore, due to such a relative robustness, besides the reduction of the number of hyperparameters of the RL algorithm that need to be set to exact values, the proposed method satisfactorily addresses the issue number 2 presented in the Introduction chapter of this work: RL algorithms usually require the adjustment of many sensitive hyperparameters, what hinders its successful use in the problem of parameter control. Despite some of the hyperparameters of TD3 were still manually set in these experiments, this is an important step towards the full hyperparameter control of algorithms of this kind.

5.2.2 Analysis of different budgets in the training and testing phases

As described in Chapter 4, one of the observed variables that describes the state of the metaheuristic (*i.e.* the state of the environment) is the ration between the elapsed number of iterations and the episode budget, expressed in percentage. Such a variable is intended to give the controller the ability to take actions according to how much is left from the initial budget for the current episode. Given that the percentage value is always relative to the total budget, and not an absolute value such as the number of elapsed iterations, the agent is supposed to make correct decisions regardless of the difference between the budgets allocated for the training and testing phases. Surely, it is expected that the closer the budgets to each other, the better, since the trained policy would be applied to a scenario it was previously trained for.

This section is intended to assess the ability of our method to apply the knowledge acquired throughout a training process executed with a budget of 100 iterations per episode to solve testing problems with a budget of 300 iterations per episode. This could be very useful to significantly reduce the training time by reducing the episode length.

Table 11 shows the number of comparisons between the $p * 100\%$ best policies for each tested budget per training episode, where they outperformed another setup and scored 0.05 or less of p-value computed through Mann-Whitney U test (all quantiles from 1.0 to 0.01) or Wilcoxon Rank-Sum test (only comparisons with the best policy). It is important to mention that each setup was compared with the other one in 29 different functions.

As expected, the experimental results showed that it is recommended to use the same budget for the training and testing phases, since such a setup outperformed its alternative in almost all comparisons. However, it is important to highlight that for the more restrictive quantiles, such a difference becomes increasingly smaller, in such a way that it vanishes when only the top policies are used for the comparisons.

Figures 59 and 60 in Appendix A.6 show the transformed p-values for the comparisons between the 30% best policies trained with 100 and 300 iterations per training episode, tested in functions 1-16 and 17-30, respectively. Figures 61 and 62 provide the same comparisons but between the best policies only. Tables 20 and 21 in Appendix A.6 show the numerical results of such experiments.

Figure 27 shows some of the comparisons provided in Figures 59 and 60 in Appendix A.6, while Figure 27 exemplifies the comparisons delivered in Figures 61 and 62 in the same appendix section. Observing the aforementioned figures, it can be seen that the superiority of the experiments with the same budget for the training and testing episodes is clear in the 30% best policies. However, such a difference cannot be observed when only the best policies are compared to each other, what corroborates with the findings observed in Table 11. Nevertheless, using different budgets for training and testing phases should be avoided given the clear superiority of its alternative setup for almost every tested case. Besides, in Machine Learning, it is always recommended to provide to the

Table 11 – Number of comparisons between the $p*100\%$ best policies for each tested budget per training episode, where they outperformed another setup and scored 0.05 or less of p-value computed through Mann-Whitney U test (all quantiles from 1.0 to 0.01) or Wilcoxon Rank-Sum test (only comparisons with the best policy).

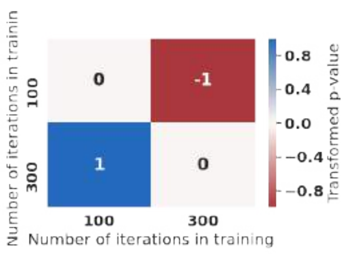
	Iterations	
Quantile (p)	**100**	**300**
1.0	4	**25**
0.9	5	**24**
0.8	3	**24**
0.7	3	**24**
0.6	4	**24**
0.5	4	**24**
0.4	5	**24**
0.3	6	**22**
0.2	9	**19**
0.1	11	**14**
0.01	5	**8**
Best	**1**	**1**

Source: Produced by the author.

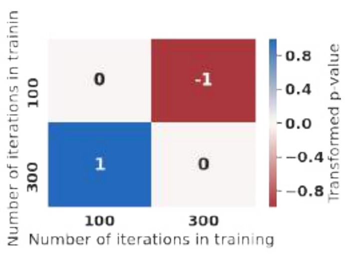
training algorithm a training condition as close as possible to the testing (*i.e.* production) environment.

Figure 27 – Sample from Figures 59 and 60 in Appendix A.6. Comparing the 30% best policies trained with 100 and 300 iterations per episode. The comparisons have been made between samples of the best fitness found in the executions of the HCLPSO algorithm. The original p-values were computed with the Mann-Whitney U test.

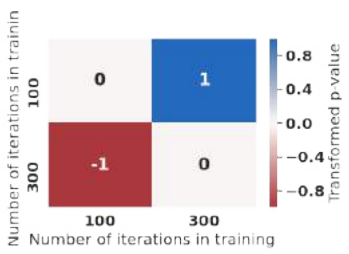
(a) Function 5



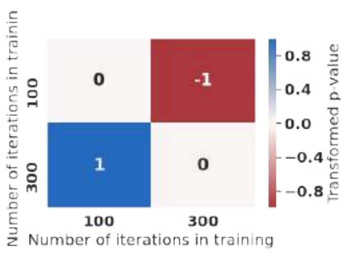
(b) Function 6



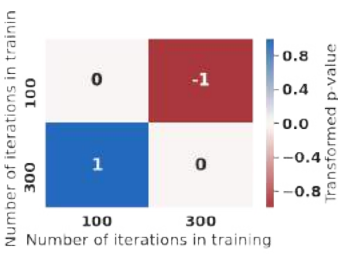
(c) Function 7



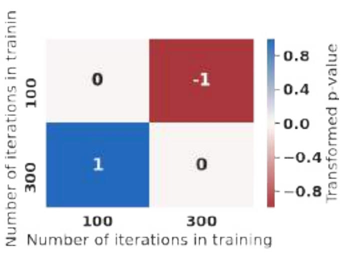
(d) Function 8



(e) Function 9



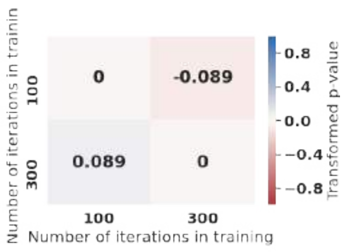
(f) Function 10



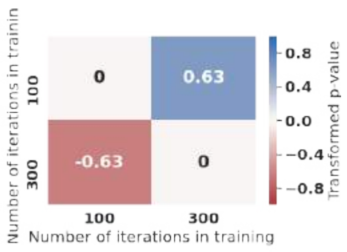
Source: Produced by the author.

Figure 28 – Sample from Figures 61 and 62 in Appendix A.6. Comparing the best policies trained with 100 and 300 iterations per episode. The comparisons have been made between samples of the best fitness found in the executions of the HCLPSO algorithm. The original p-values were computed with the Mann-Whitney U test.

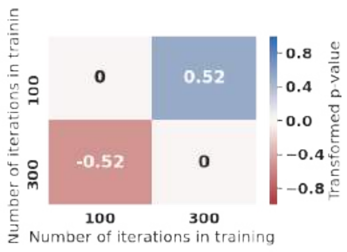
(a) Function 11



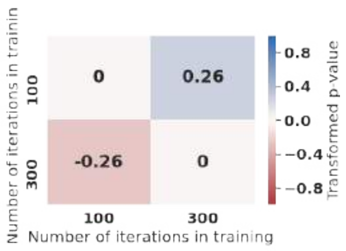
(b) Function 12



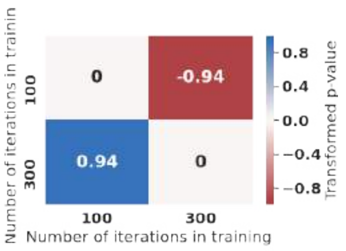
(c) Function 13



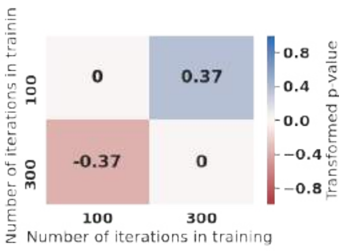
(d) Function 14



(e) Function 15



(f) Function 16



Source: Produced by the author.

Despite the aforementioned conclusions, the best policies trained in both scenarios clearly outperformed a random policy in almost all functions. It means that the knowledge acquired during a training process with 100 iterations per episode is meaningful in unseen functions with a budget of 300 iterations per episode. As previously mentioned, overcoming a random policy means that the controller left the initial condition of total ignorance about the underlying metaheuristic and the problems it solves to a trained knowledgable state. Figure 63 in Appendix A.6 shows the transformed p-values for the comparisons between the best policies trained with 100 and 300 iterations per episode with the random policy (columns 100vsR and 300vsR, respectively). Table 21 shows the numerical results of these experiments.

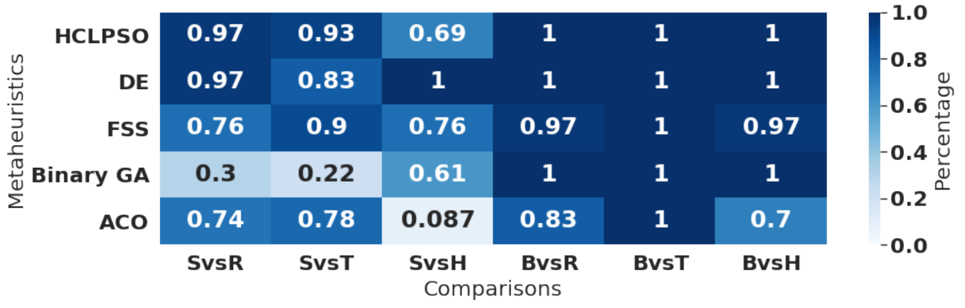
5.2.3 Generality assessment

As already mentioned, this section is intended to assess the generality of the proposed method by running the algorithm in 133 different scenarios, where 5 metaheuristics are tested in several optimization problems: HCLPSO, FSS, DE, binary GA and ACO. The results achieved by the proposed method in such experiments are compared with a random policy, with the static versions of these algorithms tuned by the I/F-Race, and with human-designed policies, which are defined by the authors of relevant papers in the literature.

Figure 29 shows the percentage of problems in which the best policy and the policy selected through the policy selection method significantly outperformed or performed similarly to the other policies. In these experiments, an algorithm A performs significantly better than an algorithm B in a given problem if the average performance of A is superior to B and the p-value of the Wilcoxon test performed with both samples is less than or equal to 0.05. Figure 30 shows the percentage of the cases where the selected and the best policies strictly significantly outperformed the other approaches. These performances were measured after 24 hours of training. SvsR, SvsT, and SvsH stand for *Selected versus Random policies*, *Selected versus Tuned static policies*, and *Selected versus Human-designed policies*, while BvsR, BvsT, and BvsH abbreviate *Best versus Random policies*, *Best versus Tuned static policies*, and *Best versus Human-designed policies.*

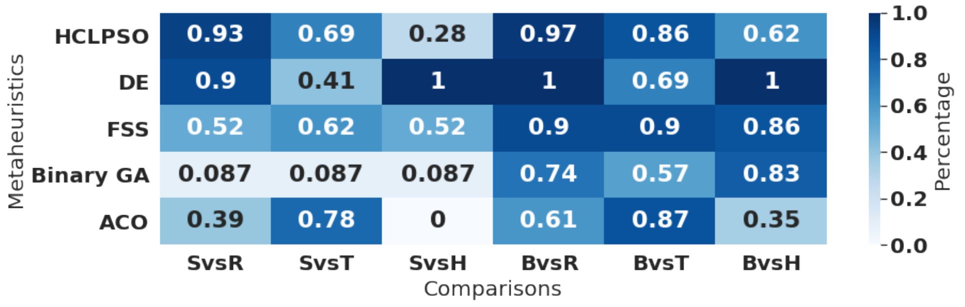
It can be seen that the best policies performed at least as well as all other approaches in at least 96% of the problems solved by HCLPSO, DE, FSS, and Binary GA, showing a very good comparative performance in ACO as well. Besides, excellent results can also be observed when only the cases where the best policies significantly overcame the other approaches are considered. In 10 out of 15 comparisons, the best trained policies outperformed the other algorithms in at least 72% of the problems. It is important to highlight that the best “machine-designed" policies performed at least as well as the human-designed policies in almost all scenarios, outperforming them in the vast majority of the tested cases. Concerning the selected policies, they performed at least as well as the other methods in the vast majority of the scenarios with HCLPSO, DE, FSS, and ACO. Moreover, despite the fact that the selected policies did not significantly outperform any of the other algorithms in almost any of the tests with Binary GA, they overcame them in most of the cases with HCLPSO, DE, and FSS. Figures 64, 65, 66, 67, and 68 in Appendix A.7 provide the transformed p-values computed for each comparison. Tables 22, 23, 24, 25, 26 in Appendix A.7 present the numerical results of the aforementioned experiments for further analysis.

Figure 29 – Percentage of problems in which the selected and the best policies performed similarly or significantly better than the other approaches (*i.e.* average performance is superior and p-value$\leq$ 0.05, or p-value$>$ 0.05). SvsR, SvsT, and SvsH refer to the comparisons between the selected trained policy and the random, the tuned, and the human-designed policies, respectively. BvsR, BvsT, and BvsH refer to the comparisons between the best trained policy and the random, the tuned, and the human-designed policies, respectively.



Source: Produced by the author.

Figure 30 – Percentage of problems in which the selected and the best policies performed significantly better than the other approaches (*i.e.* average performance is superior and p-value$\leq$ 0.05). SvsR, SvsT, and SvsH refer to the comparisons between the selected trained policy and the random, the tuned, and the human-designed policies, respectively. BvsR, BvsT, and BvsH refer to the comparisons between the best trained policy and the random, the tuned, and the human-designed policies, respectively.



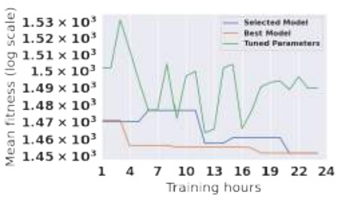
Source: Produced by the author.

It is clear that the performances of the selected policies are quite below the performance of the best policies for FSS, binary GA, and ACO. The average quantile of the selected policies for HCLPSO, DE, FSS, binary GA, and ACO in the ranked pool of trained policies is 0.042, 0.111, 0.189, 0.16, and 0.344, respectively. In other words, on average, the selected policies for HCLPSO, DE, FSS, binary GA, and ACO are placed among the 4.21%, 11.12%, 18.89%, 16.01% and 34.46% best policies of the pool, respectively. The worst performance of the selection method in the policy selection for ACO can be explained by the the fact that the instances of TSP used in this benchmark set were randomly generated, as previously mentioned. Thus, the TSP instances are highly uncorrelated, which means that finding a policy that performs well in a given set of training functions does not guarantee that it will perform well in an unseen problem instance. Despite that all selected policies are, on average, among the best policies in the pool, it can be noticed that the selection mechanism has chosen the policies with the lowest rankings for the metaheuristics that showed the worst performances. Given the excellent results presented by the best policies, it can be concluded that the proposed method is able to find very good policies for almost all 133 scenarios, but its success depends on an effective policy selection method. The quantiles of the selected policies for each metaheuristic and each problem is provided in Figure 79 in Appendix A.7.

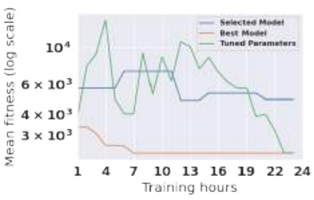
Figures 69, 70, 71, 72, 73, 74, 75, 76, 77, and 78 in Appendix A.7 show the mean best fitness found by HCLPSO, DE, FSS, binary GA, and ACO with their parameters controlled by a selected policy, the best policy and static policies tuned by I/F-Race at different points in the training time. The mean best fitnesses are calculated across the 30 executions of the metaheuristics performed with each testing function. Figure 31 shows some of the results provided by the aforementioned figures.

Figure 31 – Sample from figures 69, 70, 71, 72, 73, 74, 75, 76, 77, and 78 in Appendix A.7. Mean fitness found by HCLPSO, DE, FSS, binary GA and ACO with the selected policies, the best policies, and static tuned parameters, after some time of training in hours.

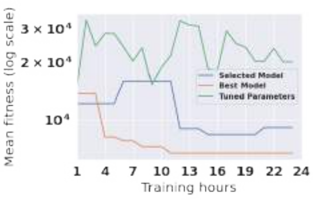
(a) Function 14 - HCLPSO



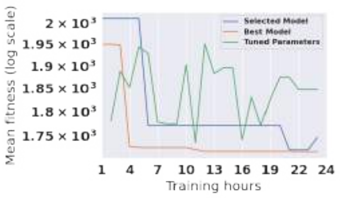
(b) Function 15 - HCLPSO



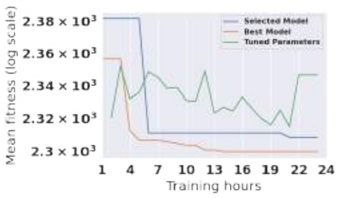
(c) Function 18 - HCLPSO



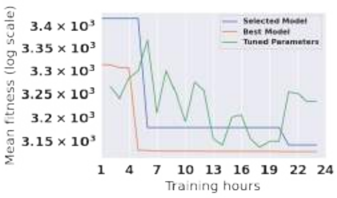
(d) Function 17 - DE



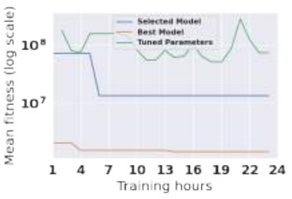
(e) Function 21 - DE



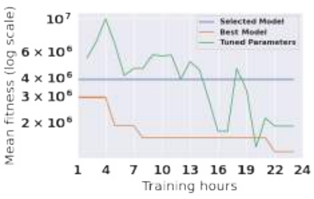
(f) Function 29 - DE



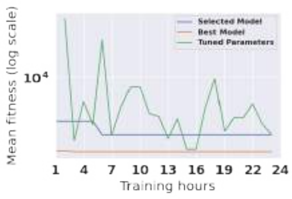
(g) Function 1 - FSS



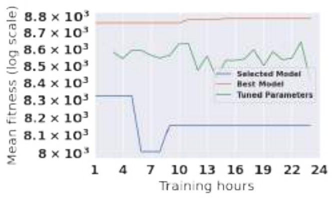
(h) Function 12 - FSS



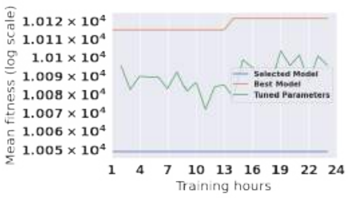
(i) Function 19 - FSS



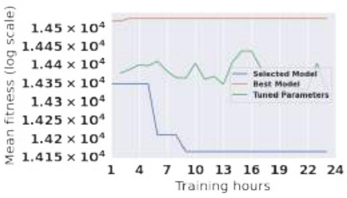
(j) Function 1 - Binary GA



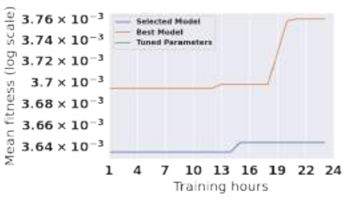
(k) Function 5 - Binary GA



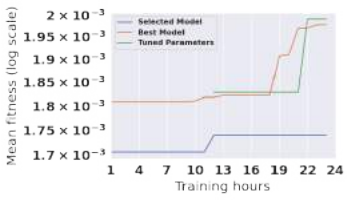
(l) Function 20- Binary GA



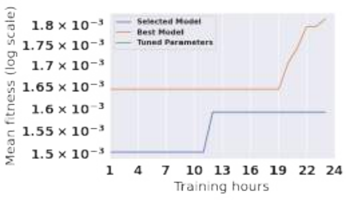
(m) Function 6 - ACO



(n) Function 15 - ACO



(o) Function 17 - ACO



Source: Produced by the author.

For the experiments with HCLPSO, DE, and FSS, the performance of the selected policies and the tuned static parameters are close to each other during most of the training time in the majority of the benchmark functions. However, in the end of the training process the final selected policy usually overcomes the static set of parameter values that survives the evolutionary pressure of the I/F-Race algorithm. Regarding the best policy, it outperforms the tuned parameters during the entire training process in almost every case. Besides, it usually shows a much better performance since the beginning of the training process.

Concerning the experiments with binary GA, it is clear that the best policy consistently outperformed the tuned parameters during the entire training process in almost every case. However, the evolution of the selected policies did not work as expected. It means that the policy selection method has failed on the selection of a good policy among the pool of trained policies. It seems that the selected policies consistently led the metaheuristic to forbidden solutions. Finally, regarding the experiments with ACO, the I/F-Race could not finish its first training iteration within 24 hours in many cases. Besides, the policy selection method showed some difficulty in selecting a top policy among the available trained policies for each experiment. Once again, the best policies showed the best performances by far in the vast majority of the cases during the entire training process. Therefore, it can be concluded that the proposed training process quickly finds very good policies. However, a more effective policy selection method should be proposed in order to identify such policies. Such a method would dramatically reduce the necessary time to find a satisfactorily good policy during the training process.

One might question the unstable curve of the tuned parameters. Such instability is due to the fact that the I/F-Race only guarantees the survival of the best parameter values ever by testing them on a set of training functions in which the unseen function is not included. Therefore, the best parameter values for the training functions may not be the best parameter values for the testing instance. The same happens to the performance of the selected model. The curve of the best model is the only one that monotonically decreases (for minimization problems) or increases (for maximization problems). This is due to the fact that, as previously explained, the best policy for each point in time is chosen by looking at the performance of all policies in the pool of trained policies in the testing function. Therefore, the best performing policy for the testing function itself is always chosen and the testing performance always increases or remains the same.

The aforementioned experiments showed that our method works for many different situations, since it was successfully tested in several numerical (*i.e.* continuous), binary, and combinatorial optimization problems, which were solved by very different metaheuristics. Therefore, it has been shown that the proposed approach has a satisfactory degree of generality. Besides, despite other approaches are claimed to be out-of-the-box, this is the study that tested the proposed method in the largest and most diverse benchmark set by far, which truly assesses the generality of the technique. Therefore, the issue number 3 found in the literature was satisfactorily addressed.

6 CONCLUSION

"And so you touch this limit, something happens and you suddenly can go a little bit further. With your mind power, your determination, your instinct, and the experience as well, you can fly very high."

Ayrton Senna

6.1 FINAL REMARKS AND MAIN CONTRIBUTIONS

This study proposed an out-of-the-box policy training method for parameter control of mono-objective EA and swarm-based algorithms with distributed Reinforcement Learning, addressing the aforementioned issues identified in the literature. Among other secondary objectives, this work was intended to address the following three gaps found in the literature:

1. Training EA and SI parameter control policies with RL can be very computational-demanding.
2. RL algorithms usually require the adjustment of many hyperparameters, what makes difficult its successful use. Also, the search for an optimal policy can be very unstable.
3. Very limited benchmarks have been used to assess the generality of the out-of-the-box methods proposed so far in the literature.

After delving into the selected problem and performing the experiments using the put forward ideas of this thesis, we argue that some novel contributions to the field of out-of-the-box parameter control for EA and swarm-based algorithms were produced, namely:

1. A systematic literature review about the given subject had never been published before. Therefore, such novelty is of substantial importance, since it organizes what has been published so far and guides the interested scientific community towards the edge of the field.
2. The proposed training process is clearly able to benefit from parallel computing platforms, even though the speed-ups are not proportional to the number of added RL workers. It means that the gap number 1 found in the literature is satisfactorily addressed.

3. PBT is used to control some of the hyperparameters of TD3 and to allow a more diverse policy search through multiple parallel training processes, known as PBT workers. The hyperparameter analysis made in this work revealed a few insights regarding the adjustment of the PBT's hyperparameters, *i.e.* the hyperparameters of the last layer of control of the proposed architecture. Overall, it can be concluded that such hyperparameters must be adjusted for the maintenance of the diversity of the policy search, so that very good policies can eventually be found in the policy search space. It means that the larger the population size of PBT workers, the less likely the policy search gets stuck in local minima and, therefore, the more likely a good policy is found. In other words, larger populations of PBT workers avoid the so-called "dog chasing its own tail" effect, where bad actions are exploited, which causes the well-known instability of model-free RL algorithms.

4. Another interesting finding observed in the hyperparameter analysis is the low sensitivity of the quality of the top policies available in the pool of trained policies to the PBT's hyperparameters. It means that the better the policy selection method, the less sensitive the selected policies to such hyperparameters. Despite the fact that many of the TD3 hyperparameters still had to be manually set for these experiments, the aforementioned findings suggest that the gap number 2 was satisfactorily addressed, especially the need for a more stable training process.

5. It is important to notice that, despite the 5 hyperparameters that are needed to be adjusted, the user of the proposed method would not need to understand the many metaheuristics available in the literature and how their perparameters affect their behaviors. This is due to the fact that the PBT layer's hyperparameters are the only ones to be defined. Therefore, the only hyperparameters to be adjusted would be the same, regardless of the underlying metaheuristic, which corroborates with the conclusion that the issue number 2 was satisfactorily addressed.

6. The generality assessment showed that our method works for many different situations, since it was successfully tested in several numerical (*i.e.* continuous), binary, and combinatorial optimization problems, which were solved by very different metaheuristics. Therefore, it has been shown that the proposed approach has a satisfactory degree of generality. Besides, despite other approaches are claimed to be out-of-the-box, this is the study that tested the proposed method in the largest and most diverse benchmark set by far, which truly assesses the generality of the technique. Therefore, the issue number 3 found in the literature was satisfactorily addressed.

7. For some of the tested cases, the best policy found by the training algorithm overcame the human-designed and tuned (*i.e.* static) policies by far. It is important

to highlight that human-designed policies usually require considerable effort from its human designers to achieve a competitive performance, while our method could outperform it in the vast majority of the cases within only 24 hours of training.

8. Schuchardt *et al.* applied for the first time an RL algorithm with continuous action space in the problem of parameter control for EA and swarm-based algorithms (SCHUCHARDT; GOLKOV; CREMERS, 2019). However, their experiments were very limited in terms of the number of simultaneously controlled parameters, and the size and diversity of the benchmark set. In this work, we have advanced the study of the application of such algorithms in the problem at hand.

The proposed method is still very costly. However, it is important to observe that once the policy is trained and selected to be used in a given unseen problem, the controller is used in the production mode. In a neural network-based policy, which is the case of TD3, the production mode involves performing only feedforward operations, which is known to be very efficient. Therefore, in continued use situations one can have an efficient and very effective automatic parameter control.

6.2 AVENUES OF FUTURE RESEARCH

A few ideas are left as future work, namely:

1. The current policy selection method chooses the same policy for different unseen problems. In fact, it only takes into account the performances of the policies in the training functions. Therefore, the success of the proposed selection method depends entirely on the choice of the functions for the training set. Thus, a method for selection of training functions should be investigated. In such an approach, the training instance that shows the most similar features to the unseen problem should be chosen to train the policy. Consequently, there would be no need for training several policies for all training functions, nor testing in all of them. Thus, a successful selection method of training functions would dramatically increase the efficiency of the method, since just a single training process would be needed to be executed and the removal of the testing phase would allow speed-ups proportional to the number of RL workers. A set of features that could be extracted from the objective functions in order to compare different problem instances are the so-called Exploratory Landscape Analysis features (ELA) (MERSMANN et al., 2011).

2. As previously mentioned, the population size control was kept out of the scope of this work. The main challenge in this problem is to create a mechanism of addition candidate solutions that does not add to much noise in the search process and a removal mechanism that does not lose important information acquired by the population. Despite the advances we have made that are not included in this study, as the

paper published in the 2018 Congress of Evolutionary Computation (LACERDA et al., 2018) and another study submitted to a scientific journal, the method proposed in this work still needs to be tested on the population size of metaheuristics.

3. The current reward function was designed in a way that almost every action has non-zero return. This is done because many of the RL algorithms do not deal well with sparse reward functions. Sometimes, in parameter control for metaheuristics, an apparently wrong decision must the made in order to maximize its expected future reward, even though such an action momentarily takes the metaheuristic to a bad region in the search space of the optimization problem. Despite the presented RL algorithms are supposed to make decisions “thinking" in the long run, they actually try to approximate the expected future return just by looking one step ahead. Therefore, an algorithm that deals with long periods without any non-zero reward and is capable of accurately assigning credits for the sequence of actions taken during such periods should be tested. This way, the reward function could return a non-zero reward only when relevant improvements are made in the search process, or even in the end of an episode, when it is known the performance of the metaheuristic after a sequence of actions. One possible candidate for such an investigation is AlphaZero (SILVER et al., 2017).

4. The proposed method should be applied to operator selection as well and could be extended to multiobjective optimization.

5. In the experiments designed for this thesis, many of the hyperparameters of the RL algorithm were manually set. Therefore, instead of reducing the difficulties in the use of the underlying metaheuristic by diminishing the complexity of parameter adjustment, it may make it even harder. However, it is important to notice that, in the proposed method, the PBT algorithm is able control the entire set of hyperparameter of any RL algorithm. Thus, in the future, we plan to make new experiments where every hyperparameter of the RL algorithm is controlled by PBT. With such a experimental setup, there would be necessary to adjust only the hyperparameters of the proposed method itself.

6. As already mentioned, the purpose of the generality assessment is to assess the performance of the proposed method in very different scenarios. Thus, instead of testing our method in the most complex instances of the problems used in the experiments, which usually require more computational resources, we decided to focus on the variability of the scenarios and keep the computational requirements for the experiments the lowest possible. However, we plan to extend the experiments presented in this work by including more complex scenarios in the future.

7. More RL algorithms will be tested in the RL layer.

8. Equation 4.2 presents a transformation for objective functions of minimization problems that does not work with problems with solutions with negative fitness values. Therefore, we plan to test our method with a different transformation function (*e.g.* multiply the original fitness values by -1).

REFERENCES

AINE, S.; KUMAR, R.; CHAKRABARTI, P. P. Adaptive parameter control of evolutionary algorithms under time constraints. In: TIWARI, A.; ROY, R.; KNOWLES, J.; AVINERI, E.; DAHAL, K. (Ed.). *Applications of Soft Computing. Advances in Intelligent and Soft Computing.* Berlin, Heidelberg: Springer, 2006.

ALAGOZ, O.; HSU, H.; SCHAEFER, A. J.; ROBERTS, M. S. Markov decision processes: A tool for sequential decision making under uncertainty. *Medical Decision Making*, v. 30, n. 4, p. 474–483, 2010.

ALETI, A.; MOSER, I. Predictive parameter control. In: *Proceedings of the 13th Annual Conference on Genetic and Evolutionary Computation.* New York, NY, USA: ACM, 2011. (GECCO '11), p. 561–568. ISBN 978-1-4503-0557-0. Disponível em: <http://doi.acm.org/10.1145/2001576.2001653>.

ALETI, A.; MOSER, I. Entropy-based adaptive range parameter control for evolutionary algorithms. In: *Proceedings of the 15th Annual Conference on Genetic and Evolutionary Computation.* New York, NY, USA: ACM, 2013. (GECCO '13), p. 1501–1508. ISBN 978-1-4503-1963-8. Disponível em: <http://doi.acm.org/10.1145/2463372.2463560>.

ALETI, A.; MOSER, I. A systematic literature review of adaptive parameter control methods for evolutionary algorithms. *ACM Comput. Surv.*, ACM, New York, NY, USA, v. 49, n. 3, p. 56:1–56:35, out. 2016. ISSN 0360-0300. Disponível em: <http://doi.acm.org/10.1145/2996355>.

ALETI, A.; MOSER, I.; MEEDENIYA, I.; GRUNSKE, L. Choosing the appropriate forecasting model for predictive parameter control. *Evolutionary Computation*, v. 22, n. 2, p. 319–349, June 2014. ISSN 1063-6560.

ALETI, A.; MOSER, I.; MOSTAGHIM, S. Adaptive range parameter control. In: *2012 IEEE Congress on Evolutionary Computation.* [S.l.: s.n.], 2012. p. 1–8. ISSN 1089-778X.

ALI, N. H. A. adn M. Z.; LIANG, J. J.; QU, B. Y.; SUGANTHAN, P. N. *Problem Definitions and Evaluation Criteria for the CEC 2017 Special Session and Competition on Single Objective Bound Constrained Real-Parameter Numerical Optimization.* 2016.

APOSTOL, K. *Temporal Difference Learning.* [S.l.]: SaluPress, 2012. ISBN 6139274524.

ASKARZADEH, A. A novel metaheuristic method for solving constrained engineering optimization problems: Crow search algorithm. *Computers & Structures*, v. 169, p. 1–12, 2016.

AWAD, N. H.; ALI, M. Z.; SUGANTHAN, P. N.; LIANG, J. J.; QU, B. Y. *Problem Definitions and Evaluation Criteria for the CEC 2017 Special Session and Competition on Single Objective Real-Parameter Numerical Optimization.* [S.l.], 2016.

BAKER, J. E. Reducing bias and inefficiency in the selection algorithm. In: *Proceedings of the Second International Conference on Genetic Algorithms on Genetic Algorithms and Their Application.* USA: L. Erlbaum Associates Inc., 1987. p. 14–21. ISBN 0805801588.

BALAPRAKASH, P.; BIRATTARI, M.; STÜTZLE, T. Improvement strategies for the f-race algorithm: Sampling design and iterative refinement. In: BARTZ-BEIELSTEIN, T.; AGUILERA, M. J. B.; BLUM, C.; NAUJOKS, B.; ROLI, A.; RUDOLPH, G.; SAMPELS, M. (Ed.). *Hybrid Metaheuristics*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2007. p. 108–122. ISBN 978-3-540-75514-2.

BALAPRAKASH, P.; BIRATTARI, M.; STÜTZLE, T. Improvement strategies for the f-race algorithm: Sampling design and iterative refinement. In: *Hybrid Metaheuristics*. [S.l.: s.n.], 2007.

BERTSEKAS, D. *Nonlinear Programming*. [S.l.]: Athena Scientific, 1999.

BERTSEKAS, D. P. *Dynamic Programming and Optimal Control*. 2nd. ed. [S.l.]: Athena Scientific, 2000. ISBN 1886529094.

BEYER, H.-G. Toward a theory of evolution strategies: Self-adaptation. *Evolutionary Computation*, v. 3, n. 3, p. 311–347, 1995. Disponível em: <https://doi.org/10.1162/evco.1995.3.3.311>.

BIELZA, C.; POZO, J. A. F. del; LARRAñAGA, P. Parameter control of genetic algorithms by learning and simulation of bayesian networks — a case study for the optimal ordering of tables. *Journal of Computer Science and Technology*, v. 28, n. 4, p. 720–731, 2013.

BIRATTARI, M. Tuning metaheuristics - a machine learning perspective. In: *Studies in Computational Intelligence*. [S.l.: s.n.], 2009.

BIRATTARI, M.; STüTZLE, T.; PAQUETE, L.; VARRENTRAPP, K. A racing algorithm for configuring metaheuristics. In: *Proceedings of the 4th Annual Conference on Genetic and Evolutionary Computation*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2002. (GECCO'02), p. 11–18. ISBN 1558608788.

BIRATTARI, M.; YUAN, Z.; BALAPRAKASH, P.; STÜTZLE, T. F-race and iterated f-race: An overview. In: *Experimental Methods for the Analysis of Optimization Algorithms*. [S.l.: s.n.], 2010.

BONABEAU, E.; DORIGO, M.; THERAULAZ, G. *From Natural to Artificial Swarm Intelligence*. USA: Oxford University Press, Inc., 1999. ISBN 0195131584.

BREST, J.; MAUčEC, M. S.; BOšKOVIć, B. il-shade: Improved l-shade algorithm for single objective real-parameter optimization. In: *2016 IEEE Congress on Evolutionary Computation (CEC)*. [S.l.: s.n.], 2016. p. 1188–1195.

BREST, J.; MAUčEC, M. S.; BOšKOVIć, B. Single objective real-parameter optimization: Algorithm jso. In: *2017 IEEE Congress on Evolutionary Computation (CEC)*. [S.l.: s.n.], 2017. p. 1311–1318.

CERNY, V. Thermodynamical approach to the traveling salesman problem: An efficient simulation algorithm. *J. Optim. Theory Appl.*, Plenum Press, USA, v. 45, n. 1, p. 41–51, jan. 1985. ISSN 0022-3239. Disponível em: <https://doi.org/10.1007/BF00940812>.

CHATZINIKOLAOU, N. Coordinating evolution: An open, peer-to-peer architecture for a self-adapting genetic algorithm. In: *Enterprise Information Systems*. [S.l.: s.n.], 2011. v. 73.

CHEN, H.; LI, G.; LIAO, H. A self-adaptive improved particle swarm optimization algorithm and its application in available transfer capability calculation. In: *2009 Fifth International Conference on Natural Computation*. [S.l.: s.n.], 2009. v. 3, p. 200–205. ISSN 2157-9563.

CONSOLI, P. A.; MEI, Y.; MINKU, L. L.; YAO, X. Dynamic selection of evolutionary operators based on online learning and fitness landscape analysis. *Soft Comput.*, Springer-Verlag, Berlin, Heidelberg, v. 20, n. 10, p. 3889–3914, out. 2016. ISSN 1432-7643. Disponível em: <https://doi.org/10.1007/s00500-016-2126-x>.

CONSOLI, P. A.; MINKU, L. L.; YAO, X. Dynamic selection of evolutionary algorithm operators based on online learning and fitness landscape metrics. In: DICK, G.; BROWNE, W. N.; WHIGHAM, P.; ZHANG, M.; BUI, L. T.; ISHIBUCHI, H.; JIN, Y.; LI, X.; SHI, Y.; SINGH, P.; TAN, K. C.; TANG, K. (Ed.). *Simulated Evolution and Learning*. Cham: Springer International Publishing, 2014. p. 359–370. ISBN 978-3-319-13563-2.

DACOSTA, L.; FIALHO, A.; SCHOENAUER, M.; SEBAG, M. Adaptive operator selection with dynamic multi-armed bandits. In: *Proceedings of the 10th Annual Conference on Genetic and Evolutionary Computation*. New York, NY, USA: Association for Computing Machinery, 2008. (GECCO '08), p. 913–920. ISBN 9781605581309. Disponível em: <https://doi.org/10.1145/1389095.1389272>.

DANG, D.-C.; LEHRE, P. K. *Self-adaptation of Mutation Rates in Non-elitist Populations*. 2016.

DANTZIG, G. Maximization of a linear function of variables subject to linear inequalities. In: . [S.l.: s.n.], 1961.

DAS, S.; MULLICK, S. S.; SUGANTHAN, P. Recent advances in differential evolution – an updated survey. *Swarm and Evolutionary Computation*, v. 27, p. 1 – 30, 2016. ISSN 2210-6502.

DEAN, J.; CORRADO, G.; MONGA, R.; CHEN, K.; DEVIN, M.; MAO, M.; RANZATO, M. aurelio; SENIOR, A.; TUCKER, P.; YANG, K.; LE, Q. V.; NG, A. Y. Large scale distributed deep networks. In: PEREIRA, F.; BURGES, C. J. C.; BOTTOU, L.; WEINBERGER, K. Q. (Ed.). *Advances in Neural Information Processing Systems 25*. [S.l.]: Curran Associates, Inc., 2012. p. 1223–1231.

DOERR, B.; WITT, C.; YANG, J. *Runtime Analysis for Self-adaptive Mutation Rates*. 2018.

DONG, C.; WANG, G.; CHEN, Z.; YU, Z. A method of self-adaptive inertia weight for pso. In: *2008 International Conference on Computer Science and Software Engineering*. [S.l.: s.n.], 2008. v. 1, p. 1195–1198. ISSN null.

DORIGO, M. *Optimization, Learning and Natural Algorithms*. Tese (Doutorado) — Politecnico di Milano, Italy, 1992.

DORIGO, M.; MANIEZZO, V.; COLORNI, A. Ant system: optimization by a colony of cooperating agents. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, v. 26, n. 1, p. 29–41, 1996.

DRAKE, J. H.; KHEIRI, A.; ÖZCAN, E.; BURKE, E. K. Recent advances in selection hyper-heuristics. *European Journal of Operational Research*, v. 285, n. 2, p. 405 – 428, 2020. ISSN 0377-2217. Disponível em: <http://www.sciencedirect.com/science/article/pii/S0377221719306526>.

EBERHART, R. C. *Computational Intelligence: Concepts to Implementations.* San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2007. ISBN 1558607595.

EIBEN, A. E.; HINTERDING, R.; MICHALEWICZ, Z. Parameter control in evolutionary algorithms. *IEEE Transactions on Evolutionary Computation*, v. 3, n. 2, p. 124–141, July 1999. ISSN 1089-778X.

EIBEN, A. E.; HORVATH, M.; KOWALCZYK, W.; SCHUT, M. C. Reinforcement learning for online control of evolutionary algorithms. In: *Proceedings of the 4th International Conference on Engineering Self-organising Systems.* Berlin, Heidelberg: Springer-Verlag, 2007. (ESOA'06), p. 151–160. ISBN 978-3-540-69867-8. Disponível em: <http://dl.acm.org/citation.cfm?id=1763581.1763595>.

EIBEN, A. E.; SMIT, S. K. Evolutionary algorithm parameters and methods to tune them. In: HAMADI, Y.; MONFROY, E.; SAUBION, F. (Ed.). *Autonomous Search.* Berlin, Heidelberg: Springer, 2011.

EIBEN, A. E.; SMITH, J. E. *Introduction to Evolutionary Computing.* 2nd. ed. [S.l.]: Springer Publishing Company, Incorporated, 2015. ISBN 3662448734.

ENGELBRECHT, A. P. *Computational Intelligence: An Introduction.* 2nd. ed. [S.l.]: Wiley Publishing, 2007. ISBN 0470035617.

FEO, T. A.; RESENDE, M. G. C. A probabilistic heuristic for a computationally difficult set covering problem. *Oper. Res. Lett.*, Elsevier Science Publishers B. V., NLD, v. 8, n. 2, p. 67–71, abr. 1989. ISSN 0167-6377. Disponível em: <https://doi.org/10.1016/0167-6377(89)90002-3>.

FILHO, C. J. A. B.; NETO, F. B. de L.; LINS, A. J. C. C.; NASCIMENTO, A. I. S.; LIMA, M. P. A novel search algorithm based on fish school behavior. In: *2008 IEEE International Conference on Systems, Man and Cybernetics.* [S.l.: s.n.], 2008. p. 2646–2651.

FILHO, C. J. A. B.; NETO, F. B. de L.; LINS, A. J. C. C.; NASCIMENTO, A. I. S.; LIMA, M. P. Fish school search. In: _____. *Nature-Inspired Algorithms for Optimisation.* Berlin, Heidelberg: Springer Berlin Heidelberg, 2009. p. 261–277.

FILHO, C. J. A. B.; NETO, F. B. L.; SOUSA, M. F. C.; PONTES, M. R.; MADEIRO, S. S. On the influence of the swimming operators in the fish school search algorithm. In: *2009 IEEE International Conference on Systems, Man and Cybernetics.* [S.l.: s.n.], 2009. p. 5012–5017.

FOGEL, L.; OWENS, A. J.; WALSH, M. J. Artificial intelligence through simulated evolution. In: . [S.l.: s.n.], 1966.

FORTNOW, L. The status of the p versus np problem. *Commun. ACM*, Association for Computing Machinery, New York, NY, USA, v. 52, n. 9, p. 78–86, set. 2009. ISSN 0001-0782. Disponível em: <https://doi.org/10.1145/1562164.1562186>.

FOULDS, L. The heuristic problem-solving approach. *Journal of the Operational Research Society*, Springer, v. 34, p. 927–934, 1983.

FUJIMOTO, S.; HOOF, H. van; MEGER, D. *Addressing Function Approximation Error in Actor-Critic Methods*. 2018.

GLOVER, F. Future paths for integer programming and links to artificial intelligence. *Comput. Oper. Res.*, Elsevier Science Ltd., GBR, v. 13, n. 5, p. 533–549, maio 1986. ISSN 0305-0548. Disponível em: <https://doi.org/10.1016/0305-0548(86)90048-1>.

GLOVER, F.; LAGUNA, M. *Tabu Search*. Norwell, MA, USA: Kluwer Academic Publishers, 1997. ISBN 079239965X.

GOLDBERG, D. E. *Genetic Algorithms in Search, Optimization and Machine Learning*. 1st. ed. USA: Addison-Wesley Longman Publishing Co., Inc., 1989. ISBN 0201157675.

GUAN, Y.; YANG, L.; SHENG, W. Population control in evolutionary algorithms: Review and comparison. In: *Bio-inspired Computing: Theories and Applications*. [S.l.: s.n.], 2017. p. 161–174.

HABIB, N. *Hands-On Q-Learning with Python: Practical Q-learning with OpenAI Gym, Keras, and TensorFlow*. Packt Publishing, 2019. ISBN 9781789345759. Disponível em: <https://books.google.com.br/books?id=xxiUDwAAQBAJ>.

Harrison, K. R.; Engelbrecht, A. P.; Ombuki-Berman, B. M. The sad state of self-adaptive particle swarm optimizers. In: *2016 IEEE Congress on Evolutionary Computation (CEC)*. [S.l.: s.n.], 2016. p. 431–439. ISSN null.

HASSELT, H. van; GUEZ, A.; SILVER, D. *Deep Reinforcement Learning with Double Q-learning*. 2015.

HESSEL, M.; MODAYIL, J.; HASSELT, H. van; SCHAUL, T.; OSTROVSKI, G.; DABNEY, W.; HORGAN, D.; PIOT, B.; AZAR, M.; SILVER, D. *Rainbow: Combining Improvements in Deep Reinforcement Learning*. 2017.

HOLLAND, J. Genetic algorithms and the optimal allocation of trials. *SIAM J. Comput.*, v. 2, p. 88–105, 1973.

HOLLAND, J. H. *Adaptation in Natural and Artificial Systems*. Ann Arbor, MI: University of Michigan Press, 1975. Second edition, 1992.

HOLLAND, J. H. *Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control and Artificial Intelligence*. Cambridge, MA, USA: MIT Press, 1992. ISBN 0262082136.

HORGAN, D.; QUAN, J.; BUDDEN, D.; BARTH-MARON, G.; HESSEL, M.; HASSELT, H. van; SILVER, D. *Distributed Prioritized Experience Replay*. 2018.

HRISTAKEVA, M. Solving the 0-1 knapsack problem with genetic algorithms. In: . [S.l.: s.n.], 2004.

Huang, C.; Li, Y.; Yao, X. A survey of automatic parameter tuning methods for metaheuristics. *IEEE Transactions on Evolutionary Computation*, v. 24, n. 2, p. 201–216, 2020.

HUTTER, F.; HOOS, H. H.; LEYTON-BROWN, K.; STUETZLE, T. Paramils: An automatic algorithm configuration framework. *Journal of Artificial Intelligence Research*, AI Access Foundation, v. 36, p. 267–306, Oct 2009. ISSN 1076-9757. Disponível em: <http://dx.doi.org/10.1613/jair.2861>.

ILAVARASI, K.; JOSEPH, K. S. Variants of travelling salesman problem: A survey. In: *International Conference on Information Communication and Embedded Systems (ICICES2014)*. [S.l.: s.n.], 2014. p. 1–7.

JADERBERG, M.; DALIBARD, V.; OSINDERO, S.; CZARNECKI, W. M.; DONAHUE, J.; RAZAVI, A.; VINYALS, O.; GREEN, T.; DUNNING, I.; SIMONYAN, K.; FERNANDO, C.; KAVUKCUOGLU, K. Population based training of neural networks. *CoRR*, abs/1711.09846, 2017. Disponível em: <http://arxiv.org/abs/1711.09846>.

JONG, K. A. D. *An Analysis of the Behavior of a Class of Genetic Adaptive Systems.* Tese (Doutorado) — University of Michigan, USA, 1975. AAI7609381.

JORDEHI, A. R.; JASNI, J. Parameter selection in particle swarm optimisation: a survey. *Journal of Experimental & Theoretical Artificial Intelligence*, v. 25, n. 4, p. 527–542, 2013.

KAELBLING, L. P.; LITTMAN, M. L.; MOORE, A. W. Reinforcement learning: A survey. *J. Artif. Int. Res.*, AI Access Foundation, El Segundo, CA, USA, v. 4, n. 1, p. 237–285, maio 1996. ISSN 1076-9757.

KARABOGA, D.; BASTURK, B. On the performance of artificial bee colony (abc) algorithm. *Appl. Soft Comput.*, Elsevier Science Publishers B. V., NLD, v. 8, n. 1, p. 687–697, jan. 2008. ISSN 1568-4946. Disponível em: <https://doi.org/10.1016/j.asoc.2007.05.007>.

KARAFOTIAS, G.; EIBEN, A. E.; HOOGENDOORN, M. Generic parameter control with reinforcement learning. In: *Proceedings of the 2014 Annual Conference on Genetic and Evolutionary Computation*. [S.l.: s.n.], 2014. (GECCO ’14), p. 1319–1326.

KARAFOTIAS, G.; HOOGENDOORN, M.; EIBEN, A. E. Evaluating reward definitions for parameter control. In: *Proceedings of the 2015 European Conference on the Applications of Evolutionary Computation*. [S.l.: s.n.], 2015. (EvoApplications ’15), p. 667–680.

KARAFOTIAS, G.; HOOGENDOORN, M.; EIBEN, A. E. Parameter control in evolutionary algorithms: Trends and challenges. *IEEE Transactions on Evolutionary Computation*, v. 19, n. 2, p. 167–187, April 2015. ISSN 1089-778X.

KARAFOTIAS, G.; HOOGENDOORN, M.; WEEL, B. Comparing generic parameter controllers for eas giorgos. In: *Proceedings of the 2014 IEEE Symposium Series on Computational Intelligence*. [S.l.: s.n.], 2014. (SSCI ’14), p. 16–53.

KARAFOTIAS, G.; SMIT, S. K.; EIBEN, A. E. A generic approach to parameter control. In: *Proceedings of the 2012 European Conference on the Applications of Evolutionary Computation*. [S.l.: s.n.], 2012. (EvoApplications ’12).

KARIMPANAL, T. G. *Neuro-evolutionary Frameworks for Generalized Learning Agents.* 2020.

KENNEDY, J.; EBERHART, R. C. Particle swarm optimization. In: *Proceedings of the IEEE International Conference on Neural Networks*. [S.l.: s.n.], 1995. p. 1942–1948.

KINGMA, D. P.; BA, J. *Adam: A Method for Stochastic Optimization*. 2014.

KOZA, J. R. *Genetic Programming: A Paradigm for Genetically Breeding Populations of Computer Programs to Solve Problems*. Stanford, CA, USA, 1990.

KRIZHEVSKY, A. *One weird trick for parallelizing convolutional neural networks*. 2014.

LACERDA, M. G. P. de; NETO, H. de A. A.; LUDERMIR, T. B.; KUCHEN, H.; NETO, F. B. de L. Population size control for efficiency and efficacy optimization in population based metaheuristics. In: *2018 IEEE Congress on Evolutionary Computation (CEC)*. [S.l.: s.n.], 2018. p. 1–8.

LACERDA, M. G. P. de; PESSOA, L. F. de A.; NETO, F. B. de L.; LUDERMIR, T. B.; KUCHEN, H. A systematic literature review on general parameter control for evolutionary and swarm-based algorithms. *Swarm and Evolutionary Computation*, v. 60, p. 100777, 2021. ISSN 2210-6502. Disponível em: <http://www.sciencedirect.com/science/article/pii/S2210650220304302>.

LAPAN, M. *Deep Reinforcement Learning Hands-On: Apply Modern RL Methods, with Deep Q-Networks, Value Iteration, Policy Gradients, TRPO, AlphaGo Zero and More*. [S.l.]: Packt Publishing, 2018. ISBN 1788834240.

LEUNG, S. W.; YUEN, S. Y.; CHOW, C. K. Parameter control system of evolutionary algorithm that is aided by the entire search history. *Appl. Soft Comput.*, Elsevier Science Publishers B. V., Amsterdam, The Netherlands, The Netherlands, v. 12, n. 9, p. 3063–3078, set. 2012. ISSN 1568-4946. Disponível em: <http://dx.doi.org/10.1016/j.asoc.2012.05.008>.

LI, A.; SPYRA, O.; PEREL, S.; DALIBARD, V.; JADERBERG, M.; GU, C.; BUDDEN, D.; HARLEY, T.; GUPTA, P. A generalized framework for population based training. In: *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery; Data Mining*. New York, NY, USA: Association for Computing Machinery, 2019. (KDD '19), p. 1791–1799. ISBN 9781450362016. Disponível em: <https://doi.org/10.1145/3292500.3330649>.

LIANG, E.; LIAW, R.; MORITZ, P.; NISHIHARA, R.; FOX, R.; GOLDBERG, K.; GONZALEZ, J. E.; JORDAN, M. I.; STOICA, I. *RLlib: Abstractions for Distributed Reinforcement Learning*. 2017.

LIANG, J. J.; QIN, A. K.; SUGANTHAN, P. N.; BASKAR, S. Comprehensive learning particle swarm optimizer for global optimization of multimodal functions. *IEEE Transactions on Evolutionary Computation*, v. 10, n. 3, p. 281–295, 2006.

LIASHCHYNSKYI, P.; LIASHCHYNSKYI, P. *Grid Search, Random Search, Genetic Algorithm: A Big Comparison for NAS*. 2019.

LILLICRAP, T. P.; HUNT, J. J.; PRITZEL, A.; HEESS, N.; EREZ, T.; TASSA, Y.; SILVER, D.; WIERSTRA, D. *Continuous control with deep reinforcement learning*. 2015.

Lloyd, S. Least squares quantization in pcm. *IEEE Transactions on Information Theory*, v. 28, n. 2, p. 129–137, March 1982. ISSN 0018-9448.

LYNN, N.; SUGANTHAN, P. N. Heterogeneous comprehensive learning particle swarm optimization with enhanced exploration and exploitation. *Swarm and Evolutionary Computation*, v. 24, p. 11 – 24, 2015. ISSN 2210-6502. Disponível em: <http://www.sciencedirect.com/science/article/pii/S2210650215000401>.

LóPEZ-IBáñEZ, M.; DUBOIS-LACOSTE, J.; STüTZLE, T.; BIRATTARI, M. *The irace Package: Iterated Racing for Automatic Algorithm Configuration*. [S.l.], 2011.

MA, Q.; STACHURSKI, J. *Dynamic Programming Deconstructed: Transformations of the Bellman Equation and Computational Efficiency*. 2019.

MARTIN, O.; OTTO, S. W.; FELTEN, E. W. Large-step markov chains for the traveling salesman problem. *Complex Systems*, v. 5, p. 299–326, 1991.

MATURANA, J.; SAUBION, F. On the design of adaptive control strategies for evolutionary algorithms. In: *Proceedings of the Evolution Artificielle, 8th International Conference on Artificial Evolution*. Berlin, Heidelberg: Springer-Verlag, 2008. (EA'07), p. 303–315. ISBN 3-540-79304-6, 978-3-540-79304-5. Disponível em: <http://dl.acm.org/citation.cfm?id=1793671.1793702>.

MERSMANN, O.; BISCHL, B.; TRAUTMANN, H.; PREUSS, M.; WEIHS, C.; RUDOLPH, G. Exploratory landscape analysis. In: *Proceedings of the 13th Annual Conference on Genetic and Evolutionary Computation*. New York, NY, USA: Association for Computing Machinery, 2011. (GECCO '11), p. 829–836. ISBN 9781450305570. Disponível em: <https://doi.org/10.1145/2001576.2001690>.

MICHALEWICZ, Z.; ARABAS, J. Genetic algorithms for the 0/1 knapsack problem. In: RAS, Z. W.; ZEMANKOVA, M. (Ed.). *Methodologies for Intelligent Systems*. Berlin, Heidelberg: Springer Berlin Heidelberg, 1994. p. 134–143. ISBN 978-3-540-49010-4.

MNIH, V.; KAVUKCUOGLU, K.; SILVER, D.; GRAVES, A.; ANTONOGLOU, I.; WIERSTRA, D.; RIEDMILLER, M. *Playing Atari with Deep Reinforcement Learning*. 2013.

MNIH, V.; KAVUKCUOGLU, K.; SILVER, D.; RUSU, A. A.; VENESS, J.; BELLEMARE, M. G.; GRAVES, A.; RIEDMILLER, M.; FIDJELAND, A. K.; OSTROVSKI, G.; PETERSEN, S.; BEATTIE, C.; SADIK, A.; ANTONOGLOU, I.; KING, H.; KUMARAN, D.; WIERSTRA, D.; LEGG, S.; HASSABIS, D. Human-level control through deep reinforcement learning. *Nature*, Nature Publishing Group, a division of Macmillan Publishers Limited. All Rights Reserved., v. 518, n. 7540, p. 529–533, fev. 2015. ISSN 00280836. Disponível em: <http://dx.doi.org/10.1038/nature14236>.

MOORE, A. W.; ATKESON, C. G. Prioritized sweeping: Reinforcement learning with less data and less time. *Mach. Learn.*, Kluwer Academic Publishers, USA, v. 13, n. 1, p. 103–130, out. 1993. ISSN 0885-6125. Disponível em: <https://doi.org/10.1023/A:1022635613229>.

NANNEN, V.; EIBEN, A. E. Relevance estimation and value calibration of evolutionary algorithm parameters. In: *Proceedings of the 20th International Joint Conference on Artifical Intelligence*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2007. (IJCAI'07), p. 975–980.

NEWBOLD, P. Arima model building and the time series analysis approach to forecasting. *Journal of Forecasting*, v. 2, n. 1, p. 23–35, 1983. Disponível em: <https://onlinelibrary.wiley.com/doi/abs/10.1002/for.3980020104>.

NICKABADI, A.; EBADZADEH, M. M.; SAFABAKHSH, R. A novel particle swarm optimization algorithm with adaptive inertia weight. *Appl. Soft Comput.*, Elsevier Science Publishers B. V., NLD, v. 11, n. 4, p. 3658–3670, jun. 2011. ISSN 1568-4946. Disponível em: <https://doi.org/10.1016/j.asoc.2011.01.037>.

NOCEDAL, J.; WRIGHT, S. J. *Numerical Optimization*. second. New York, NY, USA: Springer, 2006.

PANIGRAHI, B. K.; SHI, Y.; LIM, M.-H. *Handbook of Swarm Intelligence: Concepts, Principles and Applications*. 1st. ed. [S.l.]: Springer Publishing Company, Incorporated, 2011. ISBN 9783642173899.

PARPINELLI, R. S.; PLICHOSKI, G. F.; SILVA, R. S. da. A review of techniques for on-line control of parameters in swarm intelligence and evolutionary computation algorithms. *International Journal of Bio-inspired Computation*, v. 13, n. 1, p. 1–17, 2019.

PISINGER, D. Where are the hard knapsack problems? *Computers & Operations Research*, v. 32, n. 9, p. 2271 – 2284, 2005. ISSN 0305-0548.

PUTERMAN, M. L. Chapter 8 markov decision processes. In: *Stochastic Models*. Elsevier, 1990, (Handbooks in Operations Research and Management Science, v. 2). p. 331–434. Disponível em: <https://www.sciencedirect.com/science/article/pii/S0927050705801720>.

QIAN, C.; BIAN, C.; YU, Y.; TANG, K.; YAO, X. Analysis of noisy evolutionary optimization when sampling fails. In: *Proceedings of the Genetic and Evolutionary Computation Conference*. New York, NY, USA: Association for Computing Machinery, 2018. (GECCO '18), p. 1507–1514. ISBN 9781450356183. Disponível em: <https://doi.org/10.1145/3205455.3205643>.

RECHENBERG, I. *Evolutionsstrategie : Optimierung technischer Systeme nach Prinzipien der biologischen Evolution*. Stuttgart-Bad Cannstatt: Frommann-Holzboog, 1973. (Problemata, 15).

ROBERTSON, D. A lightweight coordination calculus for agent systems. In: *Proceedings of the Second International Conference on Declarative Agent Languages and Technologies*. Berlin, Heidelberg: Springer-Verlag, 2005. (DALT'04), p. 183–197. ISBN 3-540-26172-9, 978-3-540-26172-8. Disponível em: <http://dx.doi.org/10.1007/11493402_11>.

ROST, A.; PETROVA, I.; BUZDALOVA, A. Adaptive parameter selection in evolutionary algorithms by reinforcement learning with dynamic discretization of parameter range. In: *Proceedings of the 2016 on Genetic and Evolutionary Computation*. [S.l.: s.n.], 2016. (GECCO '16).

SCHAUL, T.; QUAN, J.; ANTONOGLOU, I.; SILVER, D. *Prioritized Experience Replay*. 2015.

SCHUCHARDT, J.; GOLKOV, V.; CREMERS, D. *Learning to Evolve*. 2019.

SER, J. D.; OSABA, E.; MOLINA, D.; YANG, X.-S.; SALCEDO-SANZ, S.; CAMACHO, D.; DAS, S.; SUGANTHAN, P. N.; COELLO, C. A. C.; HERRERA, F. Bio-inspired computation: Where we stand and what's next. *Swarm Evol. Comput.*, v. 48, p. 220–250, 2019.

SHARMA, M.; KOMNINOS, A.; IBANEZ, M. L.; KAZAKOV, D. *Deep Reinforcement Learning Based Parameter Control in Differential Evolution.* 2019.

SHI, Y.; EBERHART, R. A modified particle swarm optimizer. In: *1998 IEEE International Conference on Evolutionary Computation Proceedings. IEEE World Congress on Computational Intelligence (Cat. No.98TH8360).* [S.l.: s.n.], 1998. p. 69–73. ISSN null.

SILVER, D.; HUANG, A.; MADDISON, C. J.; GUEZ, A.; SIFRE, L.; DRIESSCHE, G. van den; SCHRITTWIESER, J.; ANTONOGLOU, I.; PANNEERSHELVAM, V.; LANCTOT, M.; DIELEMAN, S.; GREWE, D.; NHAM, J.; KALCHBRENNER, N.; SUTSKEVER, I.; LILLICRAP, T.; LEACH, M.; KAVUKCUOGLU, K.; GRAEPEL, T.; HASSABIS, D. Mastering the game of Go with deep neural networks and tree search. *Nature*, Nature Publishing Group, v. 529, n. 7587, p. 484–489, jan. 2016.

SILVER, D.; HUBERT, T.; SCHRITTWIESER, J.; ANTONOGLOU, I.; LAI, M.; GUEZ, A.; LANCTOT, M.; SIFRE, L.; KUMARAN, D.; GRAEPEL, T.; LILLICRAP, T.; SIMONYAN, K.; HASSABIS, D. *Mastering Chess and Shogi by Self-Play with a General Reinforcement Learning Algorithm.* 2017.

SILVER, E. An overview of heuristic solution methods. *Journal of the Operational Research Society*, Springer, v. 55, n. 9, p. 936–956, 2004.

STORN, R.; PRICE, K. Differential evolution – a simple and efficient heuristic for global optimization over continuous spaces. *J. of Global Optimization*, Kluwer Academic Publishers, USA, v. 11, n. 4, p. 341–359, dez. 1997. ISSN 0925-5001. Disponível em: <https://doi.org/10.1023/A:1008202821328>.

STUTZLE, T.; HOOS, H. Max-min ant system and local search for the traveling salesman problem. In: *Proceedings of 1997 IEEE International Conference on Evolutionary Computation (ICEC '97).* [S.l.: s.n.], 1997. p. 309–314.

SUTTON, R. S.; BARTO, A. G. *Reinforcement Learning: An Introduction.* Cambridge, MA, USA: A Bradford Book, 2018. ISBN 0262039249.

SUTTON, R. S.; MCALLESTER, D.; SINGH, S.; MANSOUR, Y. Policy gradient methods for reinforcement learning with function approximation. In: *Proceedings of the 12th International Conference on Neural Information Processing Systems.* Cambridge, MA, USA: MIT Press, 1999. (NIPS'99), p. 1057–1063.

TALBI, E.-G. *Metaheuristics: From Design to Implementation.* [S.l.]: Wiley Publishing, 2009. ISBN 0470278587.

TANABE, R.; FUKUNAGA, A. Success-history based parameter adaptation for differential evolution. In: *2013 IEEE Congress on Evolutionary Computation.* [S.l.: s.n.], 2013. p. 71–78.

TANABE, R.; FUKUNAGA, A. S. Improving the search performance of shade using linear population size reduction. In: *2014 IEEE Congress on Evolutionary Computation (CEC)*. [S.l.: s.n.], 2014. p. 1658–1665.

TESAURO, G. Practical issues in temporal difference learning. *Mach. Learn.*, Kluwer Academic Publishers, USA, v. 8, n. 3–4, p. 257–277, maio 1992. ISSN 0885-6125.

TONG, L.; DONG, M.; JING, C. An improved multi-population ensemble differential evolution. *Neurocomputing*, v. 290, p. 130 – 147, 2018. ISSN 0925-2312. Disponível em: <http://www.sciencedirect.com/science/article/pii/S0925231218301735>.

TOSCANI, L. V.; VELOSO, P. A. S. *Complexidade de Algoritmos*. third. [S.l.]: Bookman, 2012.

VASUKI, A. *Nature-Inspired Optimization Algorithms*. first. New York, NY, USA: Chapman and Hall/CRC, 2020.

VOUDOURIS, C. Guided local search — an illustrative example in function optimisation. *BT Technology Journal*, Kluwer Academic Publishers, USA, v. 16, n. 3, p. 46–50, jul. 1998. ISSN 1358-3948. Disponível em: <https://doi.org/10.1023/A:1009665513140>.

WANG, G.-G.; DEB, S.; COELHO, L. d. S. Elephant herding optimization. In: *Proceedings of the 2015 3rd International Symposium on Computational and Business Intelligence (ISCBI)*. USA: IEEE Computer Society, 2015. (ISCBI '15), p. 1–5. ISBN 9781467385015. Disponível em: <https://doi.org/10.1109/ISCBI.2015.8>.

WANG, Z.; SCHAUL, T.; HESSEL, M.; HASSELT, H. van; LANCTOT, M.; FREITAS, N. de. *Dueling Network Architectures for Deep Reinforcement Learning*. 2015.

WOLPERT, D. H.; MACREADY, W. G. No free lunch theorems for optimization. *IEEE Transactions on Evolutionary Computation*, v. 1, n. 1, p. 67–82, 1997.

XU, G. An adaptive parameter tuning of particle swarm optimization algorithm. *Appl. Math. Comput.*, Elsevier Science Inc., USA, v. 219, n. 9, p. 4560–4569, jan. 2013. ISSN 0096-3003. Disponível em: <https://doi.org/10.1016/j.amc.2012.10.067>.

Zhan, Z.; Zhang, J.; Li, Y.; Chung, H. S. Adaptive particle swarm optimization. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, v. 39, n. 6, p. 1362–1381, Dec 2009. ISSN 1941-0492.

ZHANG, J.; CHEN, W.-N.; ZHAN, Z.-H.; YU, W.-J.; LI, Y.-L.; CHEN, N.; ZHOU, Q. A survey on algorithm adaptation in evolutionary computation. *Frontiers of Electrical and Electronic Engineering*, v. 7, n. 1, p. 16–31, 2012. ISSN 1673-3584. Disponível em: <https://doi.org/10.1007/s11460-012-0192-0>.

Zhang, J.; Chung, H. S.; Lo, W. Clustering-based adaptive crossover and mutation probabilities for genetic algorithms. *IEEE Transactions on Evolutionary Computation*, v. 11, n. 3, p. 326–335, June 2007. ISSN 1941-0026.

ZHANG, J.; SANDERSON, A. C. Jade: Adaptive differential evolution with optional external archive. *IEEE Transactions on Evolutionary Computation*, v. 13, n. 5, p. 945–958, 2009.

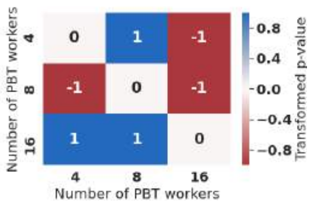
Appendix

APPENDIX A – DETAILED EXPERIMENTAL RESULTS

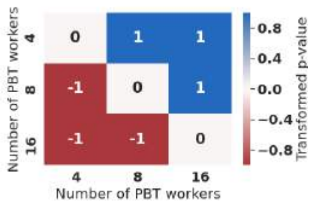
A.1 HYPERPARAMETER ANALYSIS - NUMBER OF PBT WORKERS

Figure 32 – Comparing the 30% best policies trained with different sizes of the population of PBT workers: 4, 8, and 16. The comparisons have been made between samples of the best fitness found in the executions of the HCLPSO algorithm solving functions 1-16 from the CEC17 benchmark set with the 30% best trained policies of the training set available for each function.

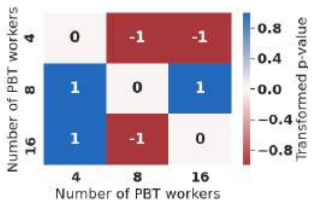
(a) Function 1



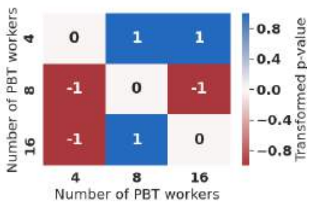
(b) Function 3



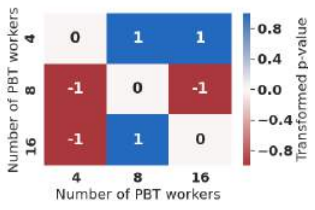
(c) Function 4



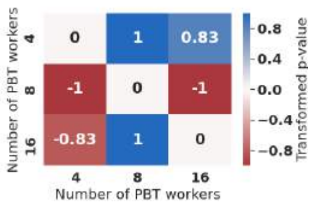
(d) Function 5



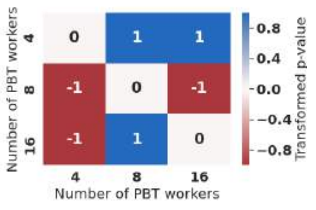
(e) Function 6



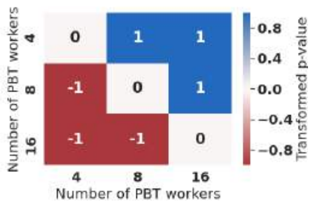
(f) Function 7



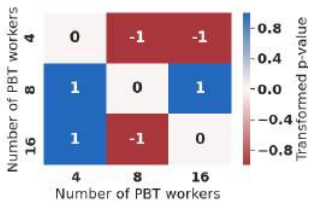
(g) Function 8



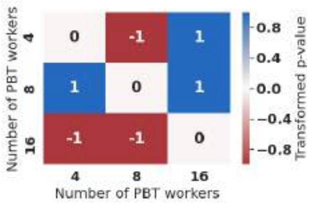
(h) Function 9



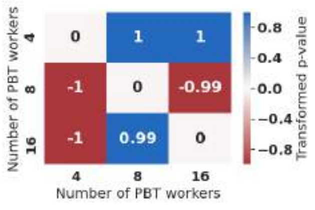
(i) Function 10



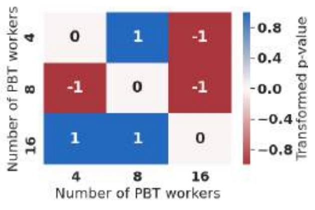
(j) Function 11



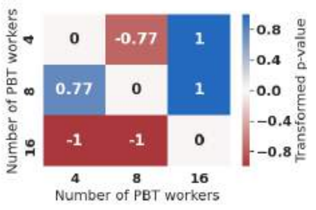
(k) Function 12



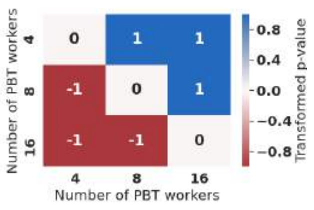
(l) Function 13



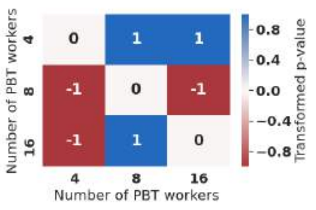
(m) Function 14



(n) Function 15



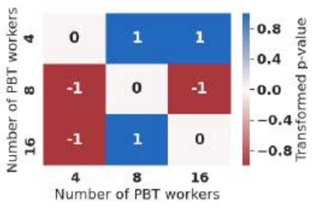
(o) Function 16



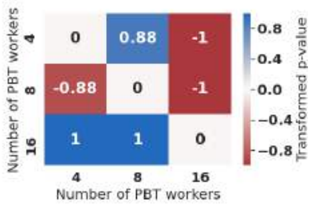
Source: Produced by the author.

Figure 33 – Comparing the 30% best policies trained with different sizes of the population of PBT workers: 4, 8, and 16. The comparisons have been made between samples of the best fitness found in the executions of the HCLPSO algorithm solving functions 17-30 from the CEC17 benchmark set with the 30% best trained policies of the training set available for each function. The original p-values were computed with the Mann-Whitney U test.

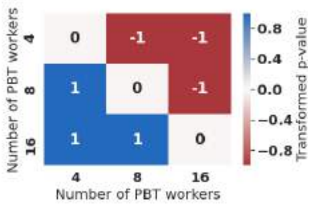
(a) Function 17



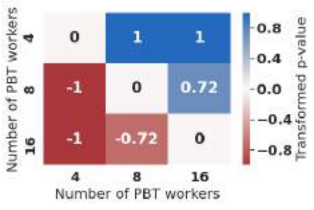
(b) Function 18



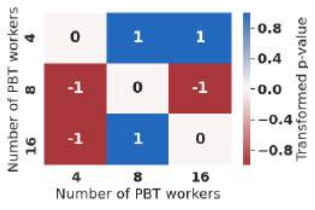
(c) Function 19



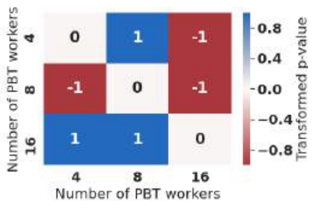
(d) Function 20



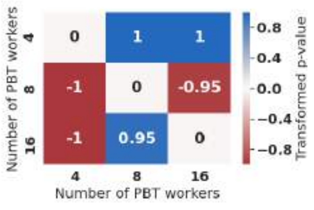
(e) Function 21



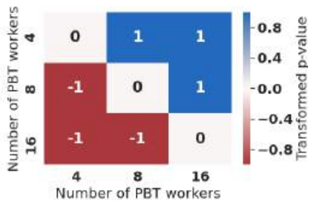
(f) Function 22



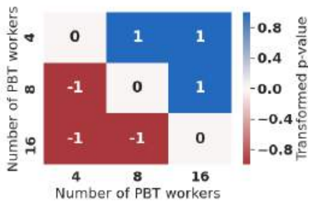
(g) Function 23



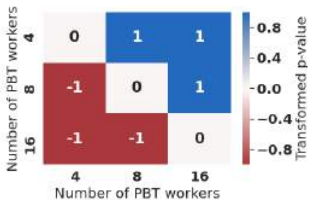
(h) Function 24



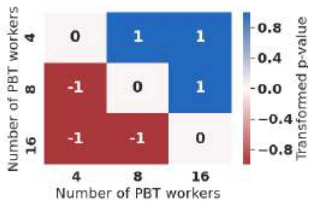
(i) Function 25



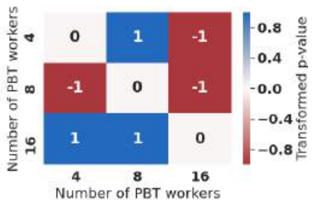
(j) Function 26



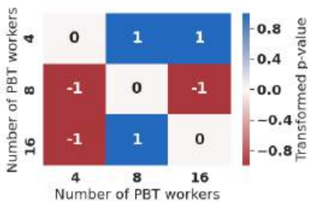
(k) Function 27



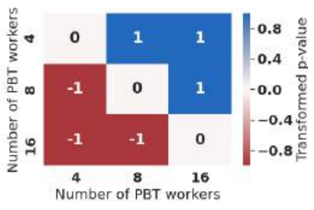
(l) Function 28



(m) Function 29



(n) Function 30



Source: Produced by the author.

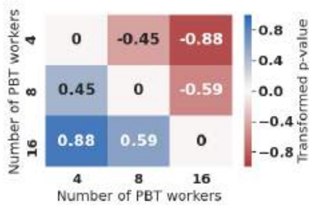
Table 12 – Mean and standard deviation (between parenthesis) of the best fitnesses found by HCLPSO with the 30% best policies trained with 4, 8, and 16 PBT workers.

Function	**4 PBT workers**	**8 PBT workers**	**16 PBT workers**
1	50153.55(113219.43)	92955.86(165974.58)	8554.90(13973.90)
3	573.42(450.80)	601.84(417.54)	782.43(791.32)
4	404.62(1.92)	404.46(1.83)	404.35(2.09)
5	523.01(8.26)	525.53(8.97)	524.57(8.79)
6	601.62(1.95)	602.70(3.16)	602.34(3.14)
7	729.30(7.78)	731.70(8.62)	729.38(7.82)
8	826.66(9.32)	829.50(9.84)	828.06(9.84)
9	935.57(78.22)	941.72(87.33)	938.38(51.90)
10	1460.37(144.13)	1437.86(130.69)	1447.53(140.47)
11	1118.36(8.85)	1117.41(8.00)	1124.43(14.24)
12	223372.22(548615.49)	197619.31(448869.89)	379037.67(894686.73)
13	3158.39(1224.48)	3292.84(1307.50)	3138.82(1310.99)
14	1470.76(17.97)	1470.41(16.94)	1473.03(17.82)
15	4147.98(2371.13)	4373.60(2444.10)	4583.84(2502.11)
16	1833.34(112.17)	1850.35(115.33)	1836.32(114.72)
17	1809.21(28.75)	1824.92(35.52)	1820.76(32.15)
18	14710.11(5612.24)	14630.16(5666.13)	13821.35(5792.08)
19	1950.98(36.30)	1947.57(33.13)	1947.12(36.83)
20	2049.44(22.70)	2052.64(26.50)	2051.86(25.23)
21	2327.62(17.61)	2330.59(16.43)	2328.93(19.57)
22	2474.70(435.08)	2758.22(599.03)	2539.31(512.17)
23	2637.54(37.53)	2646.84(41.34)	2647.51(37.22)
24	2749.57(46.47)	2757.25(32.63)	2756.13(37.49)
25	2921.56(28.70)	2924.27(29.30)	2924.44(35.14)
26	2908.55(76.84)	2913.23(62.65)	2923.27(74.02)
27	3097.48(3.83)	3098.04(4.59)	3098.24(4.12)
28	3411.42(10.57)	3411.68(5.47)	3411.58(8.19)
29	3231.86(38.08)	3239.36(40.91)	3237.72(38.13)
30	25182.87(15798.82)	26796.03(17226.01)	27340.11(18193.29)

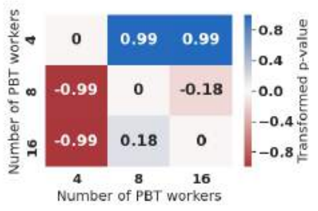
Source: Produced by the author.

Figure 34 – Comparing the best policy trained with different sizes of the population of PBT workers: 4, 8, and 16. The comparisons have been made between samples of the best fitness found in the executions of the HCLPSO algorithm solving functions 1-16 from the CEC17 benchmark set. The original p-values were computed with the Wilcoxon Rank-Sum test.

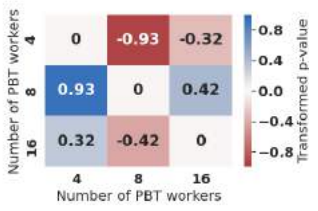
(a) Function 1



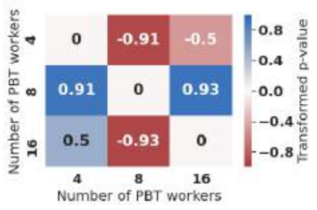
(b) Function 3



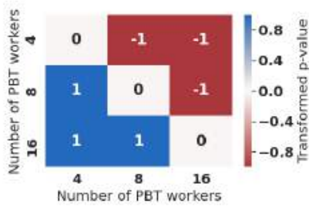
(c) Function 4



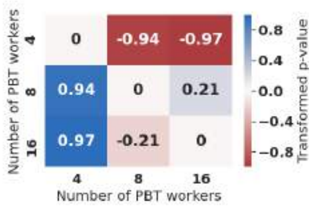
(d) Function 5



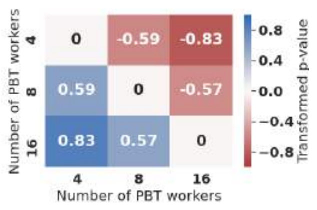
(e) Function 6



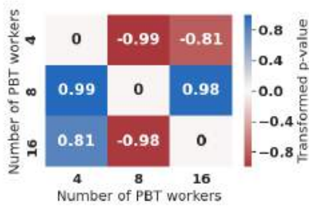
(f) Function 7



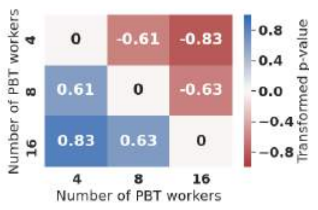
(g) Function 8



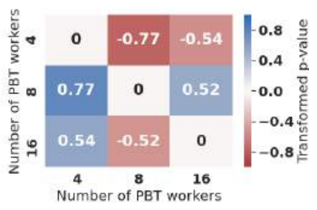
(h) Function 9



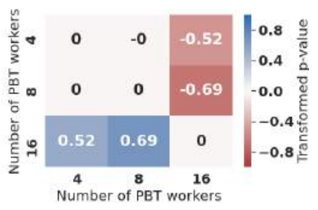
(i) Function 10



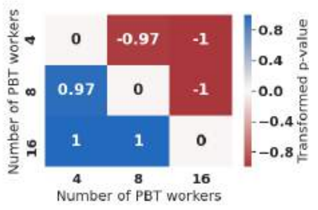
(j) Function 11



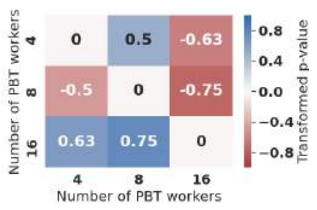
(k) Function 12



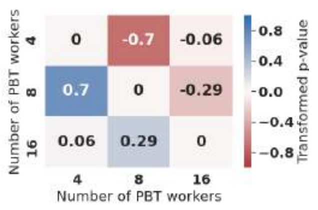
(l) Function 13



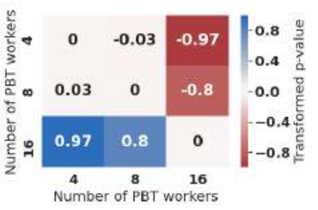
(m) Function 14



(n) Function 15



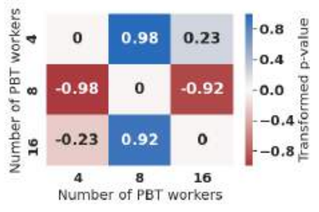
(o) Function 16



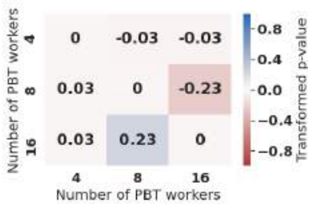
Source: Produced by the author.

Figure 35 – Comparing the best policy trained with different sizes of the population of PBT workers: 4, 8, and 16. The comparisons have been made between samples of the best fitness found in the executions of the HCLPSO algorithm solving functions 17-30 from the CEC17 benchmark set. The original p-values were computed with the Wilcoxon Rank-Sum test.

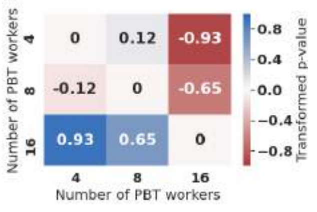
(a) Function 17



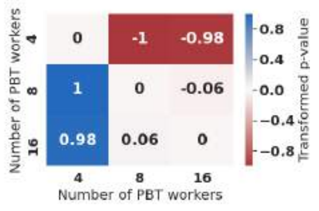
(b) Function 18



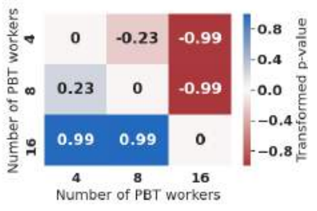
(c) Function 19



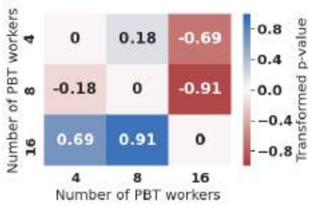
(d) Function 20



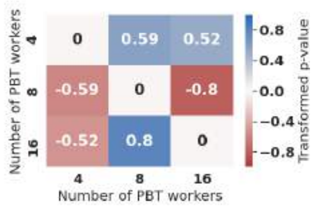
(e) Function 21



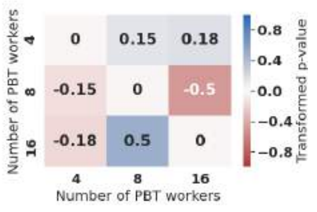
(f) Function 22



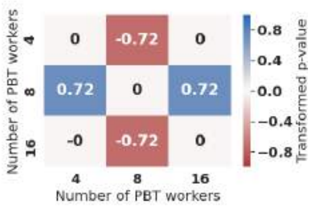
(g) Function 23



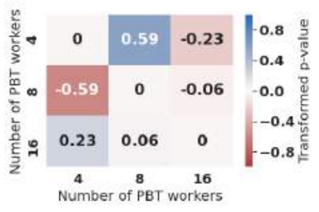
(h) Function 24



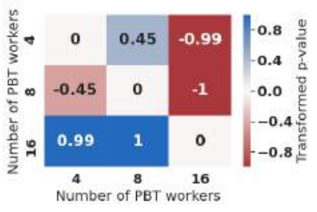
(i) Function 25



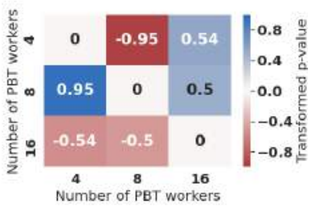
(j) Function 26



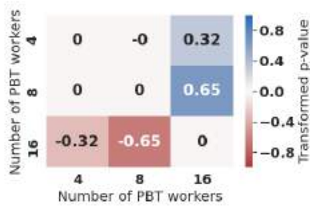
(k) Function 27



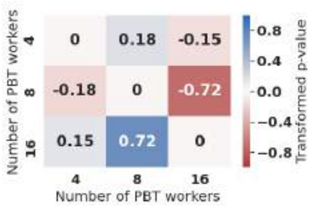
(l) Function 28



(m) Function 29



(n) Function 30



Source: Produced by the author.

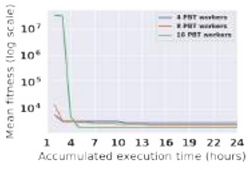
Table 13 – Mean and standard deviation (between parenthesis) of the best fitnesses found by HCLPSO with the best policy found with 4, 8, and 16 PBT workers.

Function	**4 PBT workers**	**8 PBT workers**	**16 PBT workers**
1	2744.27(2600.24)	2261.78(1571.10)	1871.58(2570.42)
3	300.48(0.90)	301.82(2.21)	302.28(2.95)
4	403.19(1.34)	402.41(1.61)	402.69(2.21)
5	512.14(5.37)	509.38(3.93)	511.32(3.45)
6	600.05(0.03)	600.01(0.01)	600.00(0.00)
7	722.29(4.13)	719.86(3.17)	719.33(3.61)
8	815.16(5.82)	813.70(5.27)	812.42(4.42)
9	900.22(0.21)	900.07(0.14)	900.12(0.08)
10	1362.72(107.01)	1335.09(70.03)	1318.66(72.49)
11	1111.95(4.87)	1110.23(5.38)	1110.61(4.36)
12	8241.11(4857.18)	8313.78(5166.85)	7183.36(5164.44)
13	2145.56(440.76)	1829.25(195.86)	1654.86(143.25)
14	1458.80(15.29)	1455.57(13.76)	1451.57(16.47)
15	2163.20(531.44)	2054.91(474.41)	2340.42(932.14)
16	1654.40(65.92)	1665.60(81.20)	1626.69(39.20)
17	1753.24(20.68)	1774.11(28.82)	1756.49(17.67)
18	6665.20(2585.54)	7075.14(3470.86)	6672.07(2674.56)
19	1920.52(17.21)	1916.80(5.96)	1915.02(9.97)
20	2026.35(8.86)	2013.99(8.10)	2016.44(13.58)
21	2271.90(42.36)	2267.90(44.81)	2228.83(39.88)
22	2238.31(20.40)	2235.68(12.24)	2229.63(13.36)
23	2576.35(104.45)	2573.66(93.05)	2556.44(126.63)
24	2655.83(82.26)	2660.49(108.84)	2651.78(74.43)
25	2888.03(96.81)	2873.30(88.30)	2862.97(132.21)
26	2803.48(137.71)	2822.31(124.97)	2826.19(188.55)
27	3093.95(2.27)	3094.21(2.01)	3092.26(1.42)
28	3392.54(58.48)	3394.52(65.68)	3380.92(44.65)
29	3191.93(21.81)	3195.47(38.34)	3199.48(19.17)
30	13619.13(7645.78)	14378.93(7159.92)	12464.23(7271.84)

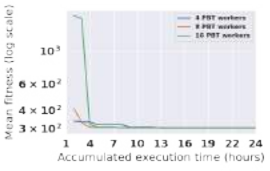
Source: Produced by the author.

Figure 36 – Mean fitness of the best policy found after some time of training in hours with 4, 8, and 16 PBT workers. Each plotted point in the lines shows the mean best fitness found by HCLPSO controlled by the best policy found so far. Only functions 1-16 are shown.

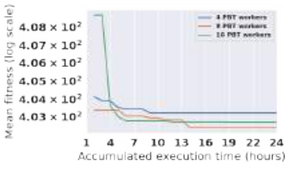
(a) Function 1



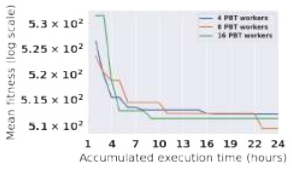
(b) Function 3



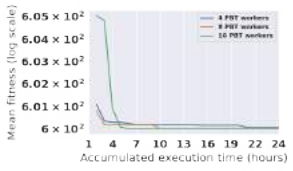
(c) Function 4



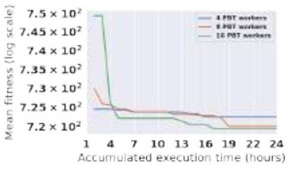
(d) Function 5



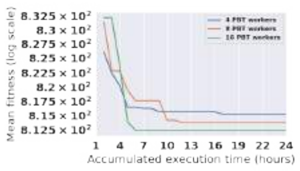
(e) Function 6



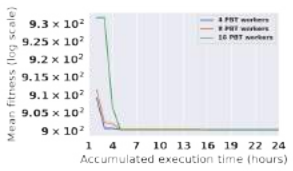
(f) Function 7



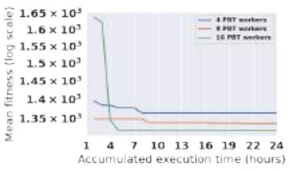
(g) Function 8



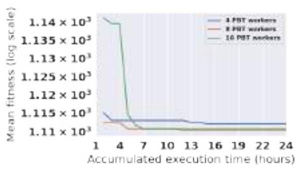
(h) Function 9



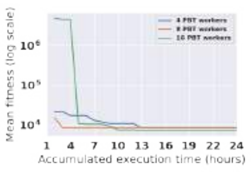
(i) Function 10



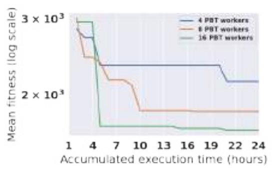
(j) Function 11



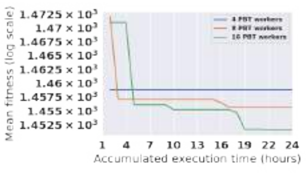
(k) Function 12



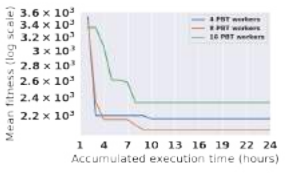
(l) Function 13



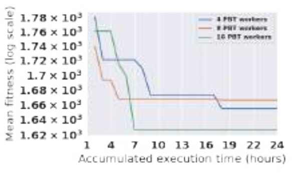
(m) Function 14



(n) Function 15



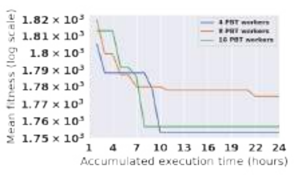
(o) Function 16



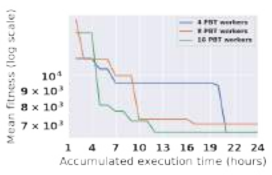
Source: Produced by the author.

Figure 37 – Mean fitness of the best policy found after some time of training in hours with 4, 8, and 16 PBT workers. Each plotted point in the lines shows the mean best fitness found by HCLPSO controlled by the best policy found so far. Only functions 17-30 are shown.

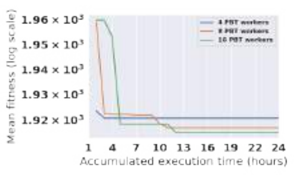
(a) Function 17



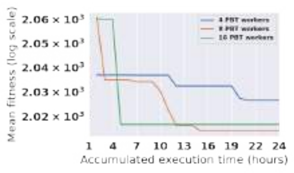
(b) Function 18



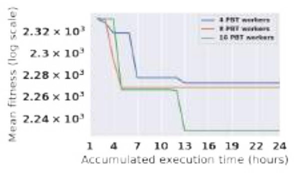
(c) Function 19



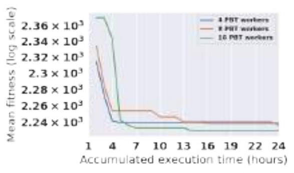
(d) Function 20



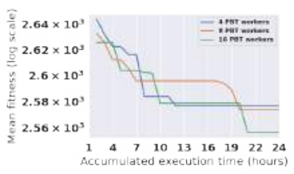
(e) Function 21



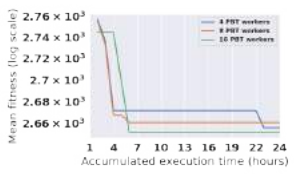
(f) Function 22



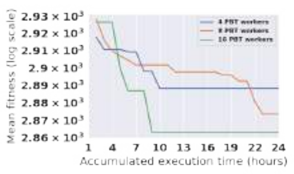
(g) Function 23



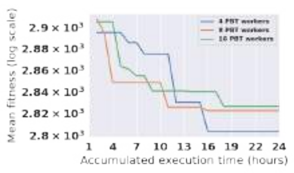
(h) Function 24



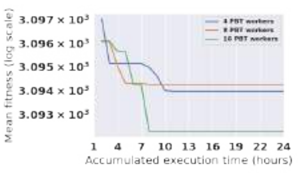
(i) Function 25



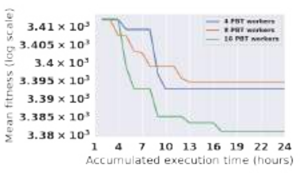
(j) Function 26



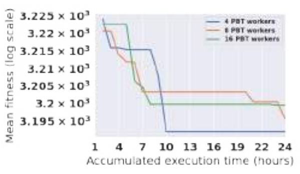
(k) Function 27



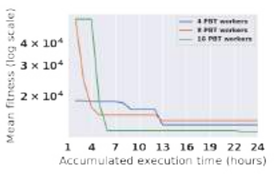
(l) Function 28



(m) Function 29



(n) Function 30

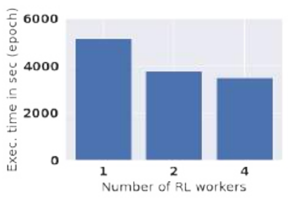


Source: Produced by the author.

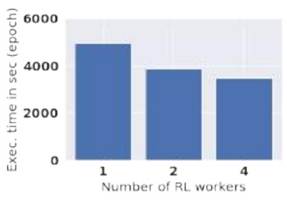
A.2 HYPERPARAMETER ANALYSIS - NUMBER OF RL WORKERS

Figure 38 – Average execution time in seconds of one training epoch during the training process with 1, 2, and 4 RL workers, for each training function (functions 1-16).

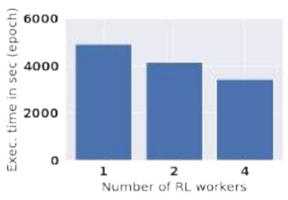
(a) Function 1



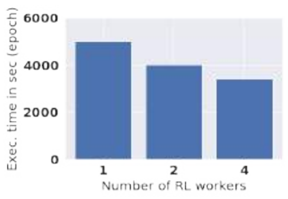
(b) Function 3



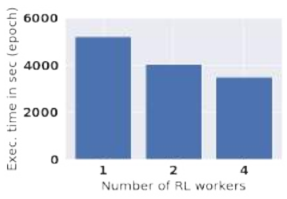
(c) Function 4



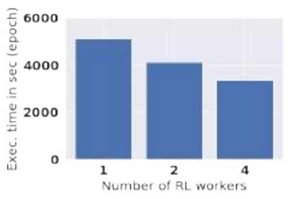
(d) Function 5



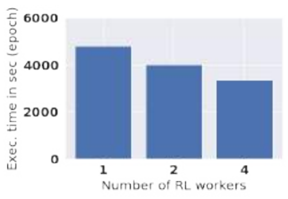
(e) Function 6



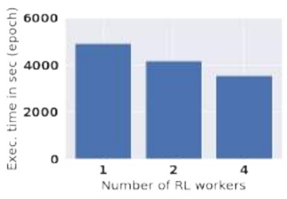
(f) Function 7



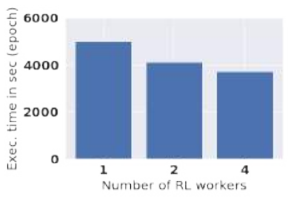
(g) Function 8



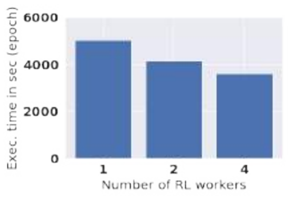
(h) Function 9



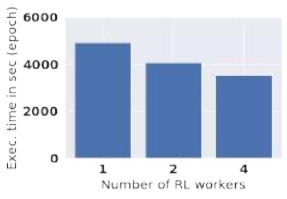
(i) Function 10



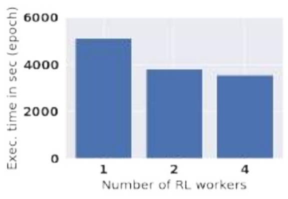
(j) Function 11



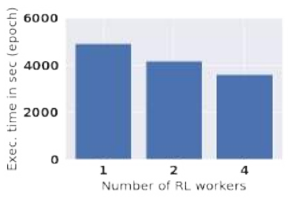
(k) Function 12



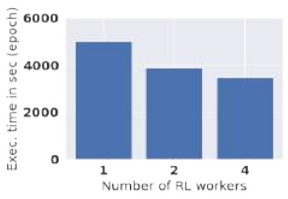
(l) Function 13



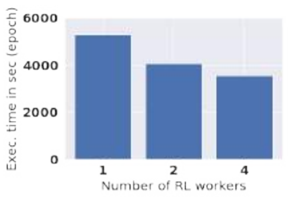
(m) Function 14



(n) Function 15

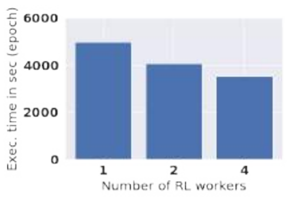


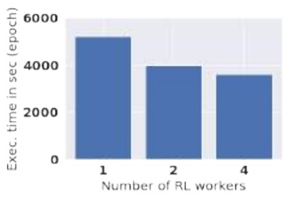
(o) Function 16

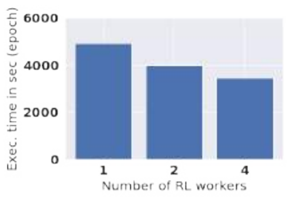


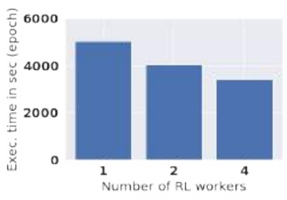
Source: Produced by the author.

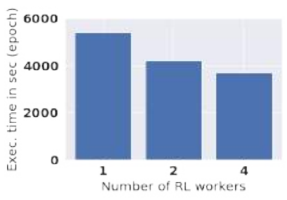
Figure 39 – Average execution time in seconds of one training epoch during the training process with 1, 2, and 4 RL workers, for each training function (functions 17-30).

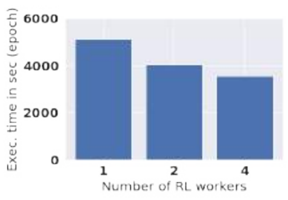

(a) Function 17

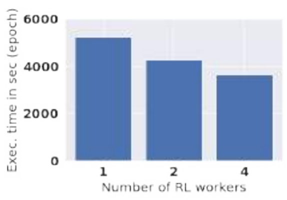

(b) Function 18

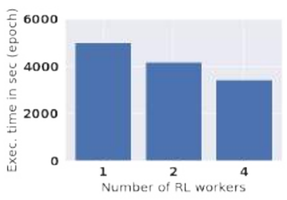

(c) Function 19

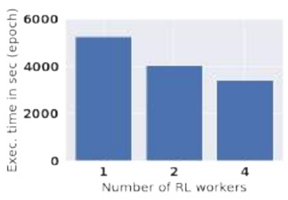

(d) Function 20

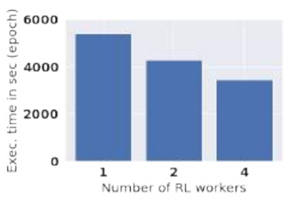

(e) Function 21

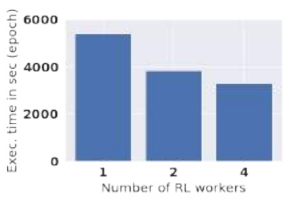

(f) Function 22

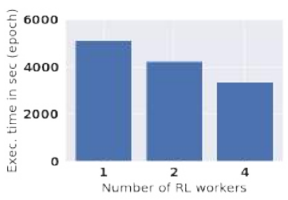

(g) Function 23

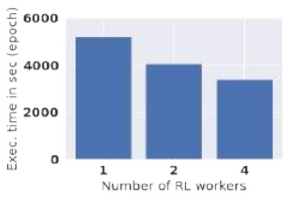

(h) Function 24

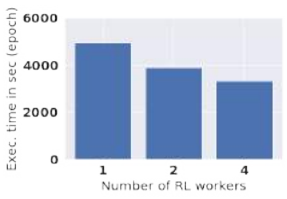

(i) Function 25


(j) Function 26


(k) Function 27


(l) Function 28


(m) Function 29

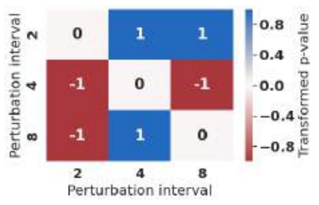

(n) Function 30

Source: Produced by the author.

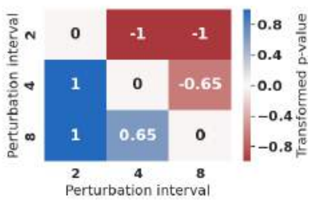
A.3 HYPERPARAMETER ANALYSIS - PERTURBATION INTERVAL

Figure 40 – Comparing the 30% best policies trained with different perturbation intervals: 2, 4, and 8 iterations. The comparisons have been made between samples of the best fitness found in the executions of the HCLPSO algorithm solving functions 1-16 from the CEC17 benchmark set. The original p-values were computed with the Mann-Whiteney U test.

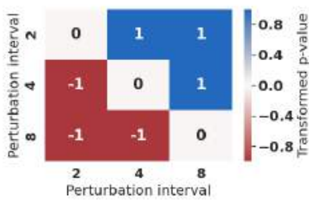
(a) Function 1



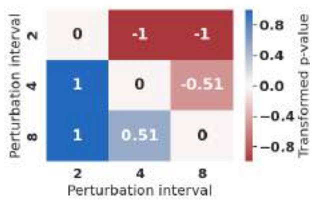
(b) Function 3



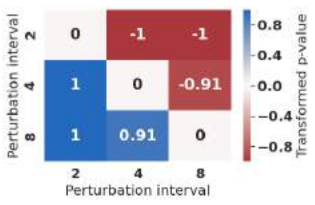
(c) Function 4



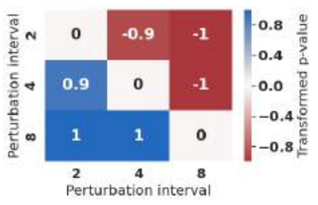
(d) Function 5



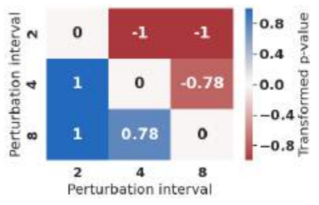
(e) Function 6



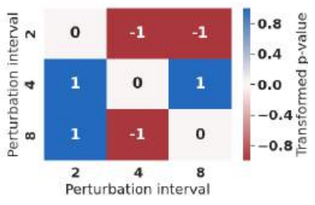
(f) Function 7



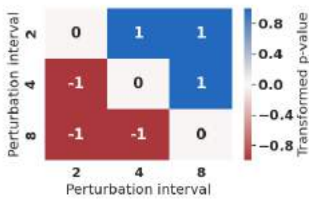
(g) Function 8



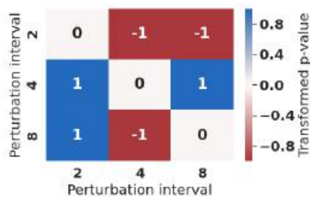
(h) Function 9



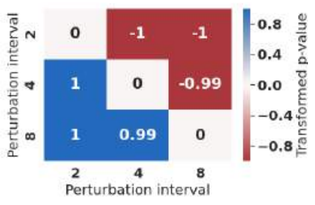
(i) Function 10



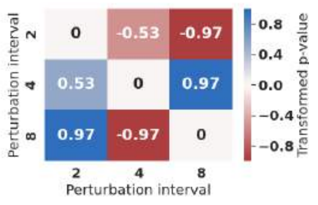
(j) Function 11



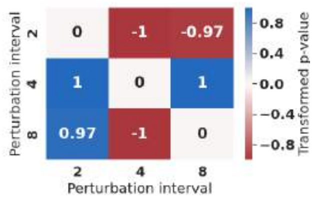
(k) Function 12



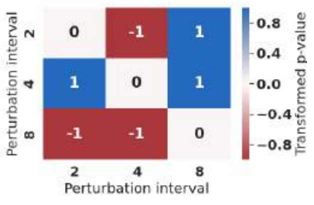
(l) Function 13



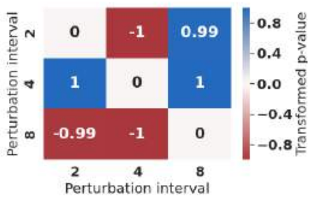
(m) Function 14



(n) Function 15



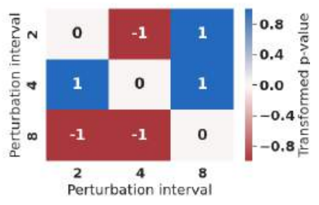
(o) Function 16



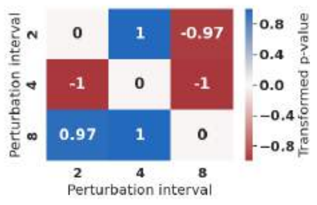
Source: Produced by the author.

Figure 41 – Comparing the 30% best policies trained with different intervals: 2, 4, and 8 iterations. The comparisons have been made between samples of the best fitness found in the executions of the HCLPSO algorithm solving functions 17-30 from the CEC17 benchmark set. The original p-values were computed with the Mann-Whiteney U test.

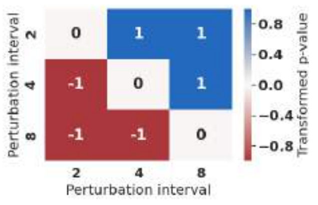
(a) Function 17



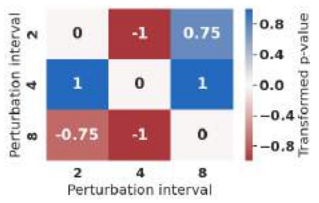
(b) Function 18



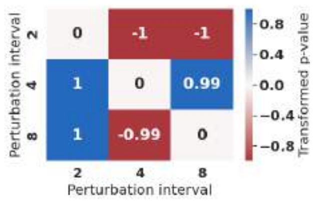
(c) Function 19



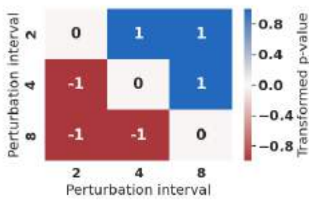
(d) Function 20



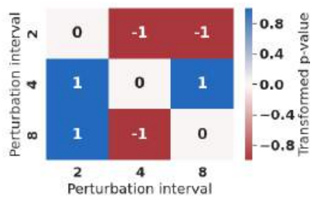
(e) Function 21



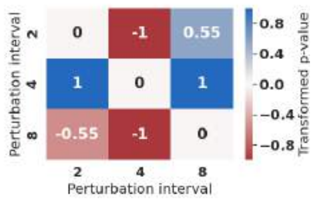
(f) Function 22



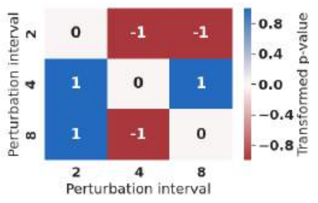
(g) Function 23



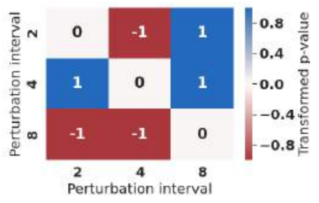
(h) Function 24



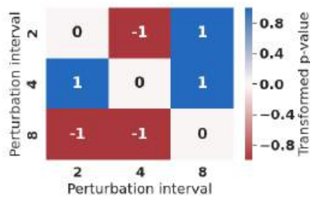
(i) Function 25



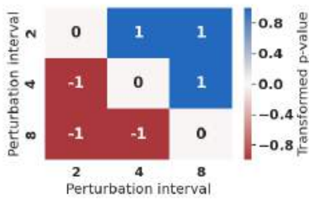
(j) Function 26



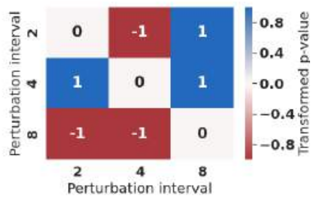
(k) Function 27



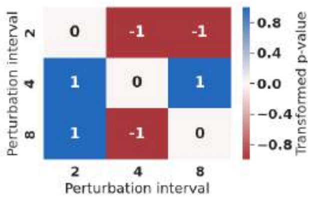
(l) Function 28



(m) Function 29



(n) Function 30



Source: Produced by the author.

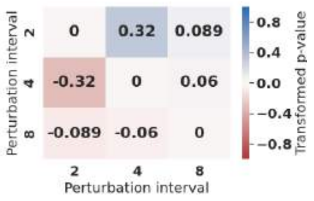
Table 14 – Mean and standard deviation (between parenthesis) of the best fitnesses found by HCLPSO with the 30% best policies trained with 2, 4 and 8 iterations of perturbation interval.

Function	**2 iterations**	**4 iterations**	**8 iterations**
1	8554.90(13973.90)	43223.09(117239.20)	40666.55(116995.93)
3	782.43(791.32)	518.48(349.13)	556.50(478.65)
4	404.35(2.09)	404.68(2.08)	404.78(2.21)
5	524.57(8.79)	523.60(8.33)	523.61(8.45)
6	602.34(3.14)	602.16(3.01)	602.39(3.33)
7	729.38(7.82)	729.34(7.80)	728.97(7.77)
8	828.06(9.84)	826.46(9.25)	826.41(9.33)
9	938.38(51.90)	942.72(90.89)	952.62(120.92)
10	1447.53(140.47)	1459.01(146.04)	1469.08(151.90)
11	1124.43(14.24)	1121.89(13.13)	1124.91(17.91)
12	379037.67(894686.73)	285201.15(723700.98)	333109.65(890314.91)
13	3138.82(1310.99)	3144.16(1314.20)	3175.08(1361.04)
14	1473.03(17.82)	1471.99(18.23)	1472.86(18.41)
15	4583.84(2502.11)	4523.43(2531.56)	4752.28(2711.13)
16	1836.32(114.72)	1832.89(113.27)	1838.62(114.66)
17	1820.76(32.15)	1815.76(30.45)	1821.51(33.05)
18	13821.35(5792.08)	14081.42(5948.50)	14006.41(6180.81)
19	1947.12(36.83)	1950.16(38.65)	1953.75(42.47)
20	2051.86(25.23)	2049.87(24.07)	2051.77(25.10)
21	2328.93(19.57)	2326.41(21.68)	2327.25(19.89)
22	2539.31(512.17)	2513.22(483.93)	2481.75(456.92)
23	2647.51(37.22)	2642.32(37.18)	2643.65(37.15)
24	2756.13(37.49)	2754.72(39.62)	2756.83(36.10)
25	2924.44(35.14)	2923.90(31.60)	2925.38(34.23)
26	2923.27(74.02)	2918.58(79.47)	2928.76(86.78)
27	3098.24(4.12)	3097.97(3.99)	3098.93(4.72)
28	3411.58(8.19)	3411.41(10.07)	3411.54(7.93)
29	3237.72(38.13)	3236.36(38.61)	3241.20(40.14)
30	27340.11(18193.29)	25643.20(16621.70)	26388.23(17735.60)

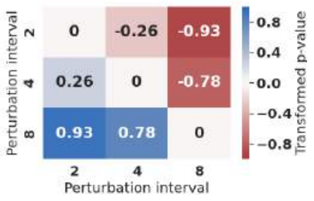
Source: Produced by the author.

Figure 42 – Comparing the best policy trained with different perturbation intervals: 2, 4, and 8 iterations. The comparisons have been made between samples of the best fitness found in the executions of the HCLPSO algorithm solving functions 1-16 from the CEC17 benchmark set. The original p-values were computed with the Wilcoxon Rank-Sum test.

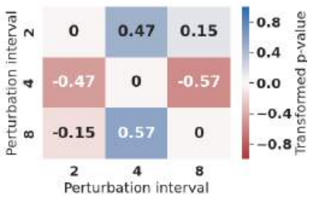
(a) Function 1



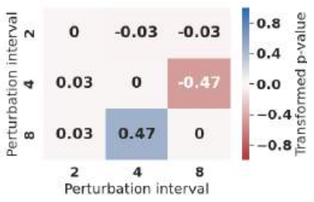
(b) Function 3



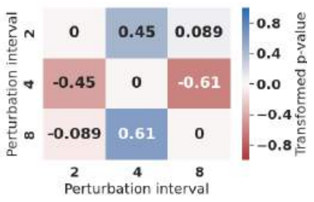
(c) Function 4



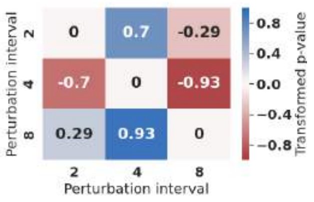
(d) Function 5



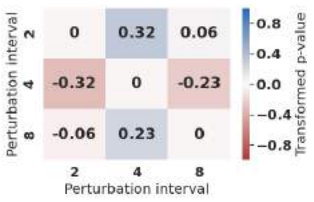
(e) Function 6



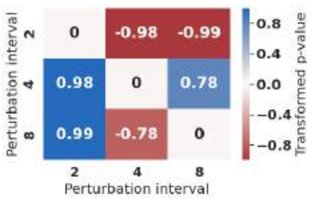
(f) Function 7



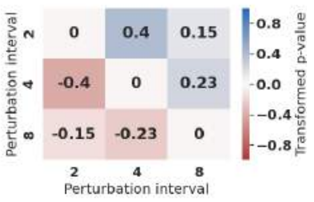
(g) Function 8



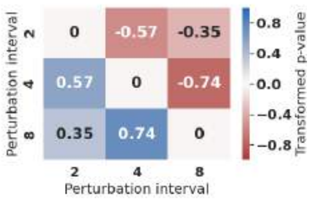
(h) Function 9



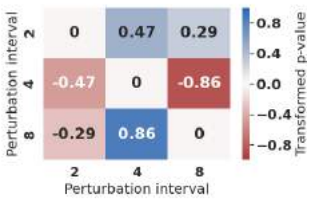
(i) Function 10



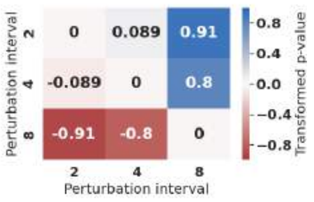
(j) Function 11



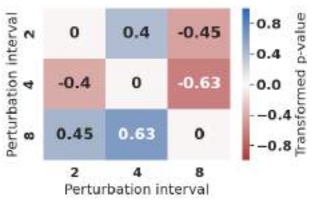
(k) Function 12



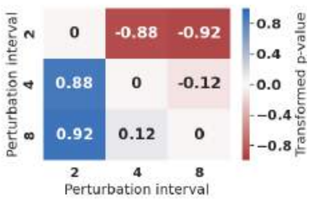
(l) Function 13



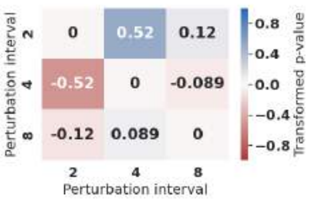
(m) Function 14



(n) Function 15



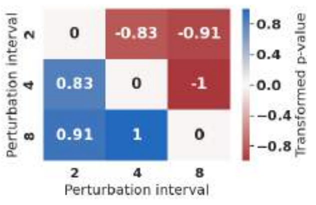
(o) Function 16



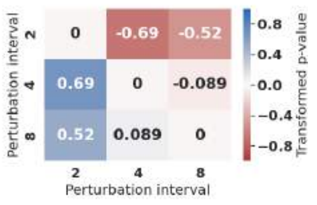
Source: Produced by the author.

Figure 43 – Comparing the best policy trained with different perturbation intervals: 2, 4, and 8 iterations. The comparisons have been made between samples of the best fitness found in the executions of the HCLPSO algorithm solving functions 17-30 from the CEC17 benchmark set. The original p-values were computed with the Wilcoxon Rank-Sum test.

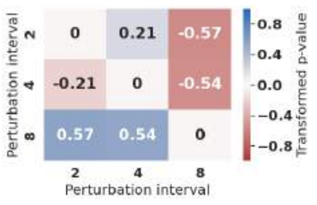
(a) Function 17



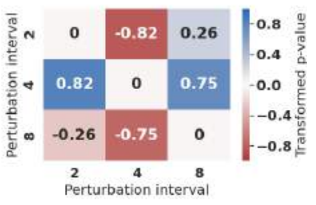
(b) Function 18



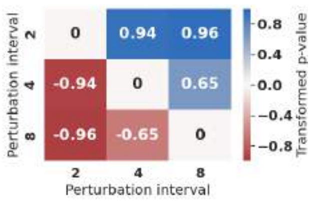
(c) Function 19



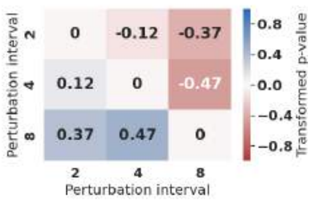
(d) Function 20



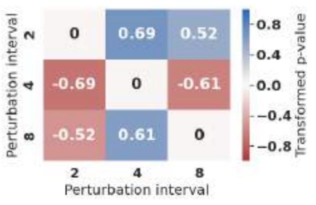
(e) Function 21



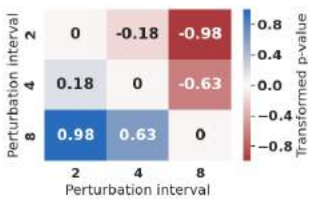
(f) Function 22



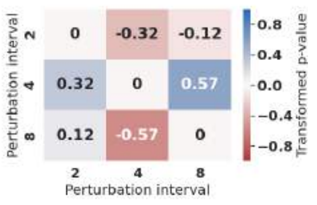
(g) Function 23



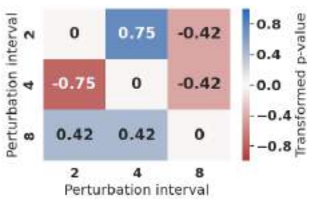
(h) Function 24



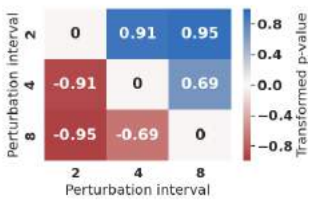
(i) Function 25



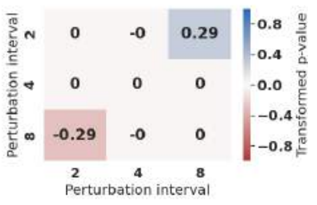
(j) Function 26



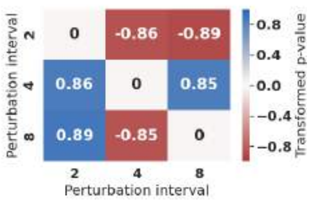
(k) Function 27



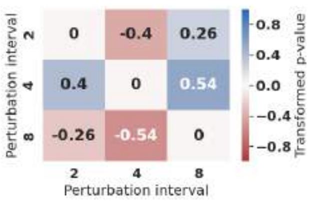
(l) Function 28



(m) Function 29



(n) Function 30



Source: Produced by the author.

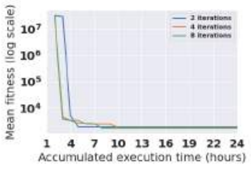
Table 15 – Mean and standard deviation (between parenthesis) of the best fitnesses found by HCLPSO with the best policy found with 2, 4 and 8 iterations of perturbation interval.

Function	**2 iterations**	**4 iterations**	**8 iterations**
1	1871.58(2570.42)	1760.98(2119.78)	1687.90(1784.74)
3	302.28(2.95)	301.53(1.41)	301.00(1.38)
4	402.69(2.21)	402.84(1.75)	402.45(1.72)
5	511.32(3.45)	511.61(3.82)	511.01(3.58)
6	600.00(0.00)	600.01(0.01)	600.00(0.00)
7	719.33(3.61)	720.71(3.58)	719.08(4.77)
8	812.42(4.42)	812.94(3.66)	812.61(4.17)
9	900.12(0.08)	900.08(0.22)	900.05(0.11)
10	1318.66(72.49)	1336.10(73.91)	1324.20(61.94)
11	1110.61(4.36)	1111.88(5.68)	1110.31(4.36)
12	7183.36(5164.44)	7734.00(4635.55)	6260.91(2744.10)
13	1654.86(143.25)	1680.41(203.28)	1770.48(241.27)
14	1451.57(16.47)	1453.18(15.51)	1448.80(12.04)
15	2340.42(932.14)	1915.48(426.31)	1845.26(294.09)
16	1626.69(39.20)	1629.79(51.49)	1636.16(45.67)
17	1756.49(17.67)	1768.19(25.58)	1747.28(10.59)
18	6672.07(2674.56)	5906.80(2353.29)	5871.19(2740.23)
19	1915.02(9.97)	1912.83(5.17)	1911.98(6.02)
20	2016.44(13.58)	2011.56(8.88)	2016.87(12.57)
21	2228.83(39.88)	2237.39(23.01)	2244.03(28.95)
22	2229.63(13.36)	2229.00(14.11)	2227.24(8.43)
23	2556.44(126.63)	2573.32(107.54)	2564.38(120.05)
24	2651.78(74.43)	2646.33(90.12)	2613.06(50.81)
25	2862.97(132.21)	2881.26(95.43)	2877.38(104.32)
26	2826.19(188.55)	2784.57(143.34)	2762.46(149.65)
27	3092.26(1.42)	3093.22(1.36)	3093.81(2.65)
28	3380.92(44.65)	3377.73(107.29)	3384.57(83.46)
29	3199.48(19.17)	3191.34(17.32)	3187.96(22.23)
30	12464.23(7271.84)	12404.88(7540.49)	12282.12(3917.91)

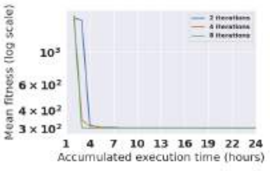
Source: Produced by the author.

Figure 44 – Mean fitness of the best policy found after some time of training in hours with perturbation intervals of 2, 4, and 8 iterations. Each plotted point in the lines shows the mean best fitness found by HCLPSO controlled by the best policy found so far. Only functions 1-16 are shown.

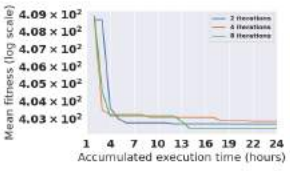
(a) Function 1



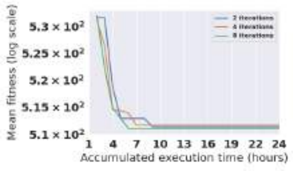
(b) Function 3



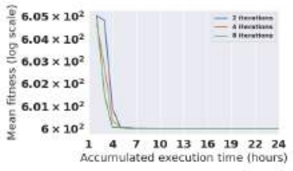
(c) Function 4



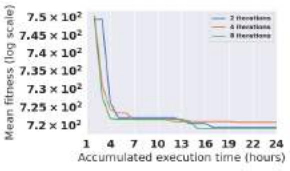
(d) Function 5



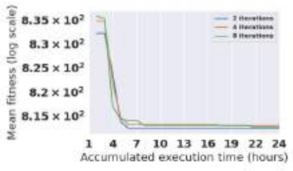
(e) Function 6



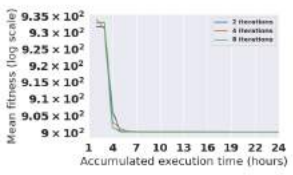
(f) Function 7



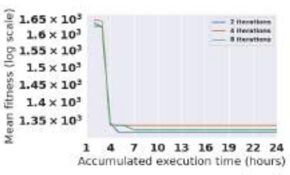
(g) Function 8



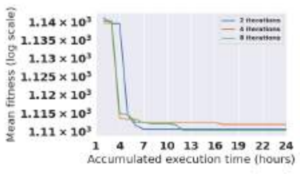
(h) Function 9



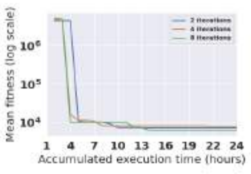
(i) Function 10



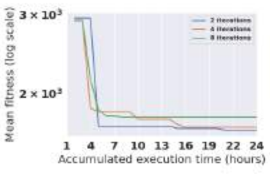
(j) Function 11



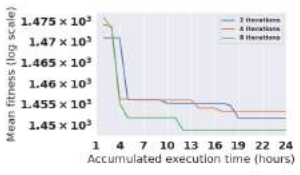
(k) Function 12



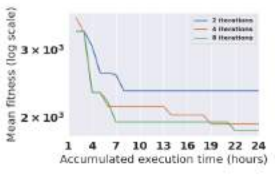
(l) Function 13



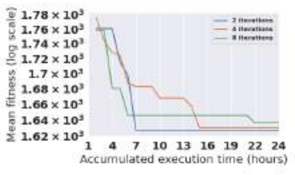
(m) Function 14



(n) Function 15



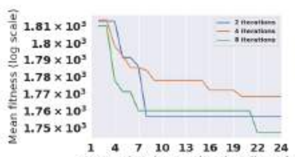
(o) Function 16



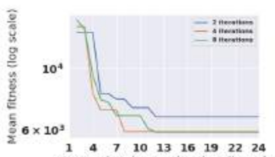
Source: Produced by the author.

Figure 45 – Mean fitness of the best policy found after some time of training in hours with perturbation intervals of 2, 4, and 8 iterations. Each plotted point in the lines shows the mean best fitness found by HCLPSO controlled by the best policy found so far. Only functions 17-30 are shown.

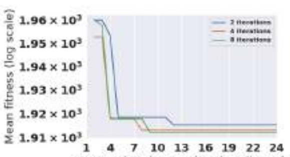
(a) Function 17



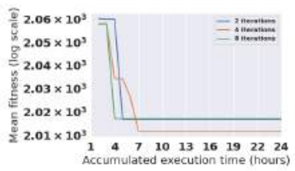
(b) Function 18



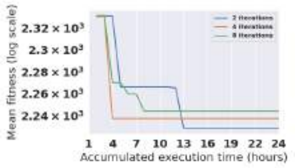
(c) Function 19



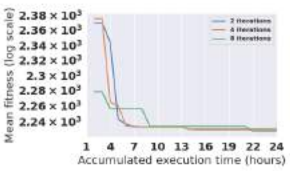
(d) Function 20



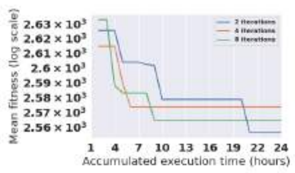
(e) Function 21



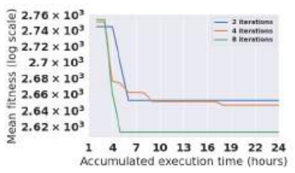
(f) Function 22



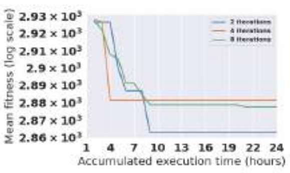
(g) Function 23



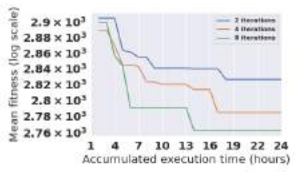
(h) Function 24



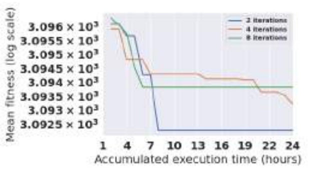
(i) Function 25



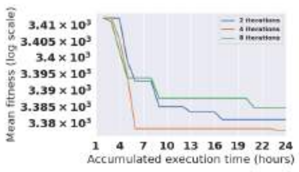
(j) Function 26



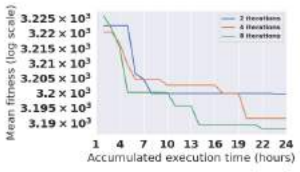
(k) Function 27



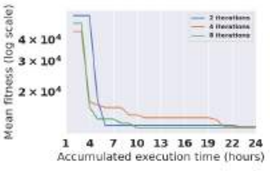
(l) Function 28



(m) Function 29



(n) Function 30

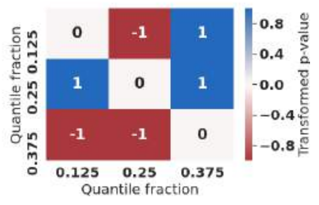


Source: Produced by the author.

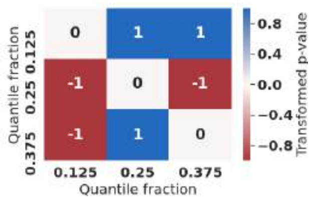
A.4 HYPERPARAMETER ANALYSIS - QUANTILE FRACTION

Figure 46 – Comparing the 30% best policies trained with different quantile fractions: 0.125, 0.25, 0.375. The comparisons have been made between samples of the best fitness found in the executions of the HCLPSO algorithm solving functions 1-16 from the CEC17 benchmark set. The original p-values were computed with the Mann-Whitney U test.

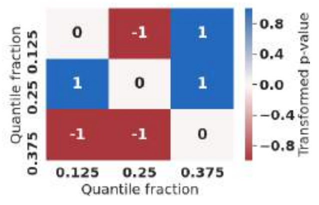
(a) Function 1



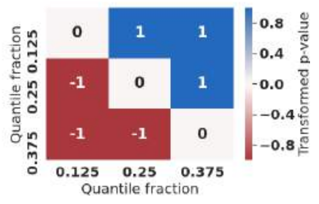
(b) Function 3



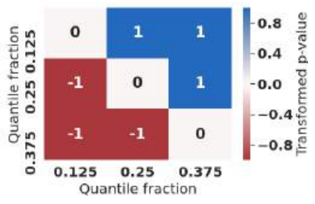
(c) Function 4



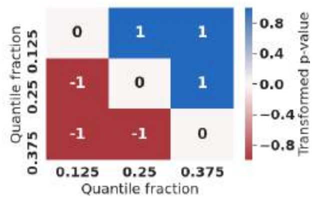
(d) Function 5



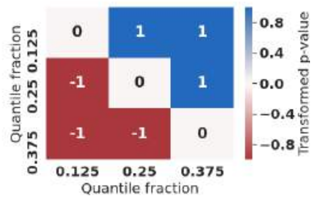
(e) Function 6



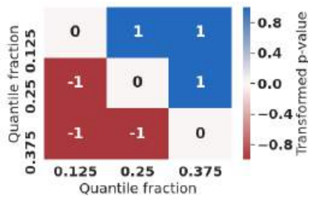
(f) Function 7



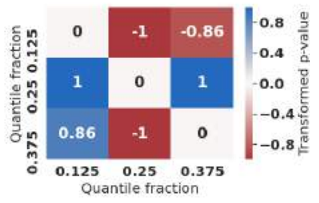
(g) Function 8



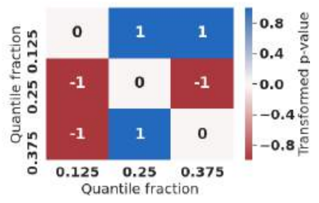
(h) Function 9



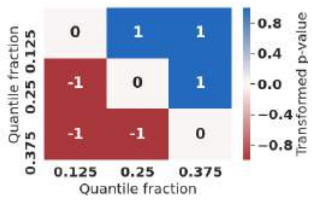
(i) Function 10



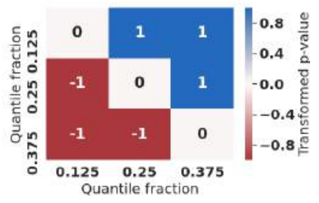
(j) Function 11



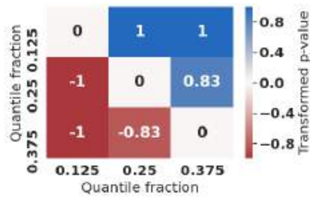
(k) Function 12



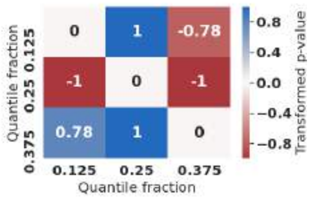
(l) Function 13



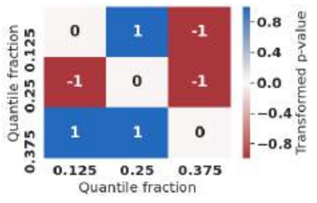
(m) Function 14



(n) Function 15



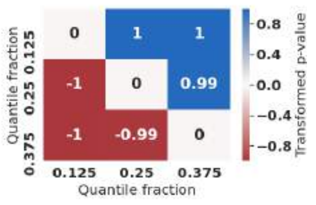
(o) Function 16



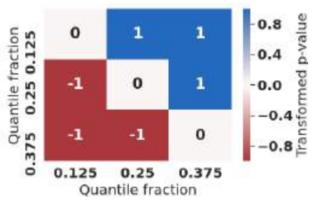
Source: Produced by the author.

Figure 47 – Comparing the 30% best policies trained with different quantile fractions: 0.125, 0.25, 0.375. The comparisons have been made between samples of the best fitness found in the executions of the HCLPSO algorithm solving functions 17-30 from the CEC17 benchmark set. The original p-values were computed with the Mann-Whitney U test.

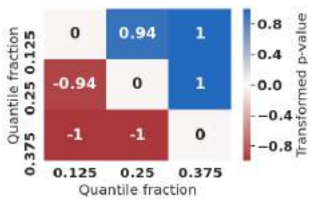
(a) Function 17



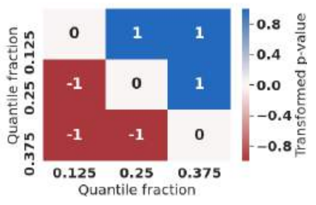
(b) Function 18



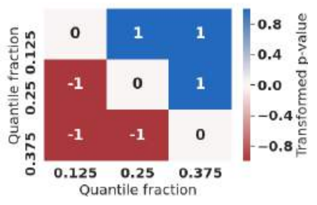
(c) Function 19



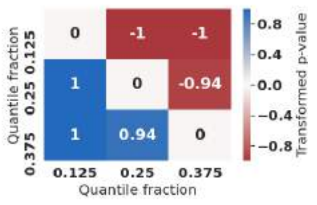
(d) Function 20



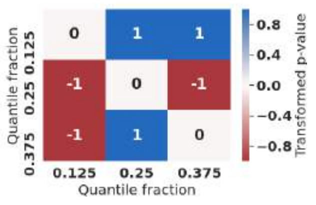
(e) Function 21



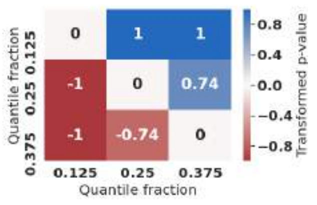
(f) Function 22



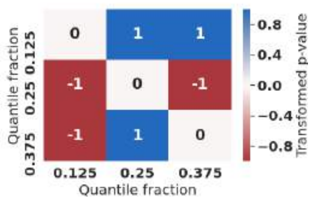
(g) Function 23



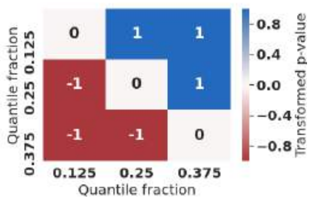
(h) Function 24



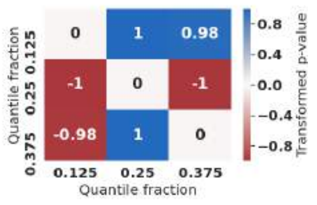
(i) Function 25



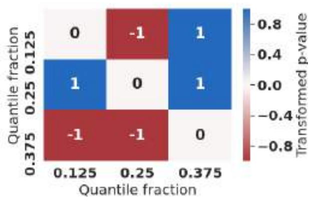
(j) Function 26



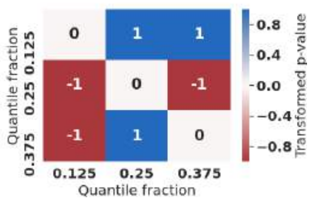
(k) Function 27



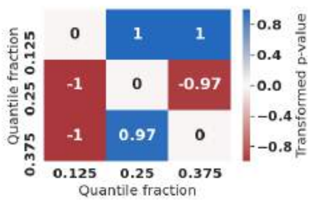
(l) Function 28



(m) Function 29



(n) Function 30



Source: Produced by the author.

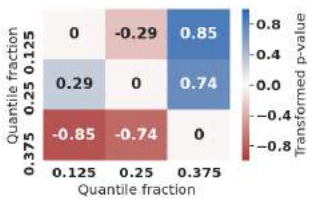
Table 16 – Mean and standard deviation (between parenthesis) of the best fitnesses found by HCLPSO with the 30% best policies trained with different quantile fractions: 0.125, 0.25, and 0.375.

Function	**Quantile: 0.125**	**Quantile: 0.25**	**Quantile: 0.375**
1	8554.90(13973.90)	37505.78(90437.82)	51867.84(105183.66)
3	782.43(791.32)	499.53(282.06)	511.77(341.93)
4	404.35(2.09)	404.40(1.95)	404.46(1.95)
5	524.57(8.79)	523.59(8.26)	523.84(8.49)
6	602.34(3.14)	601.86(2.56)	601.95(2.48)
7	729.38(7.82)	729.04(7.62)	729.35(7.76)
8	828.06(9.84)	826.67(9.23)	827.61(9.61)
9	938.38(51.90)	933.62(47.66)	930.78(36.92)
10	1447.53(140.47)	1443.67(136.58)	1445.08(138.06)
11	1124.43(14.24)	1119.25(9.48)	1119.34(9.82)
12	379037.67(894686.73)	214801.41(539974.14)	211618.46(532502.97)
13	3138.82(1310.99)	3045.63(1203.92)	2981.10(1152.46)
14	1473.03(17.82)	1470.40(17.60)	1469.75(17.36)
15	4583.84(2502.11)	4475.72(2481.99)	4488.95(2511.58)
16	1836.32(114.72)	1840.97(114.93)	1835.11(119.48)
17	1820.76(32.15)	1816.52(30.98)	1817.83(32.85)
18	13821.35(5792.08)	13769.20(5606.61)	13659.41(5519.70)
19	1947.12(36.83)	1945.81(34.07)	1944.49(32.87)
20	2051.86(25.23)	2048.42(23.46)	2049.26(23.83)
21	2328.93(19.57)	2328.44(16.67)	2328.73(15.28)
22	2539.31(512.17)	2593.18(541.45)	2576.45(535.01)
23	2647.51(37.22)	2644.23(38.16)	2643.73(40.53)
24	2756.13(37.49)	2756.51(35.74)	2756.15(35.52)
25	2924.44(35.14)	2922.42(30.99)	2923.01(30.20)
26	2923.27(74.02)	2911.61(73.64)	2910.39(70.84)
27	3098.24(4.12)	3097.86(3.81)	3097.87(3.89)
28	3411.58(8.19)	3411.57(8.02)	3411.44(9.26)
29	3237.72(38.13)	3235.81(39.29)	3234.92(38.96)
30	27340.11(18193.29)	25022.49(15927.70)	25323.89(16111.62)

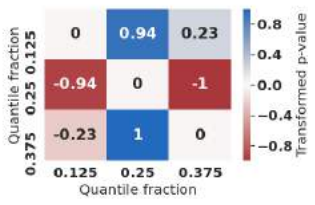
Source: Produced by the author.

Figure 48 – Comparing the best policy trained with different quantile fractions: 0.125, 0.25, 0.375. The comparisons have been made between samples of the best fitness found in the executions of the HCLPSO algorithm solving functions 1-16 from the CEC17 benchmark set. The original p-values were computed with the Wilcoxon Rank-Sum test.

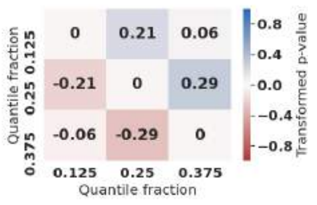
(a) Function 1



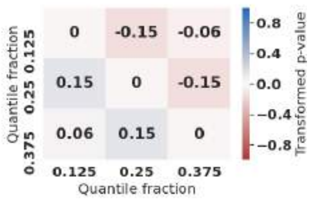
(b) Function 3



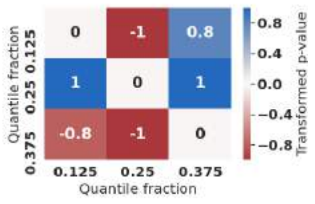
(c) Function 4



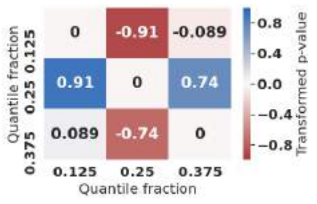
(d) Function 5



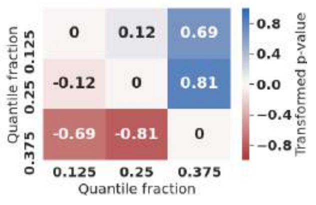
(e) Function 6



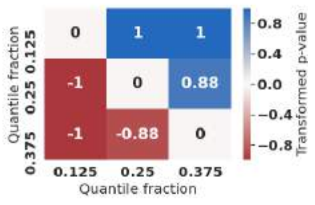
(f) Function 7



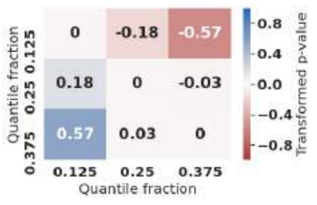
(g) Function 8



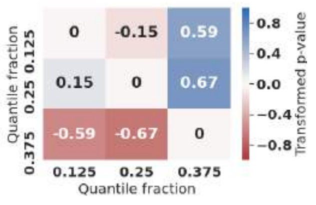
(h) Function 9



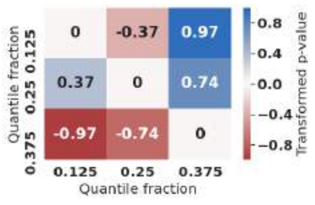
(i) Function 10



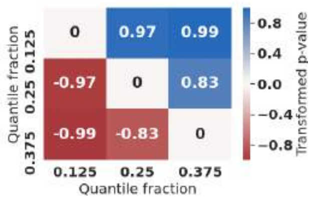
(j) Function 11



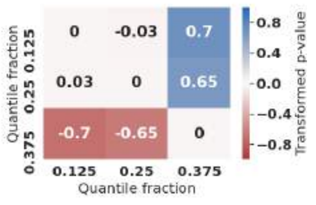
(k) Function 12



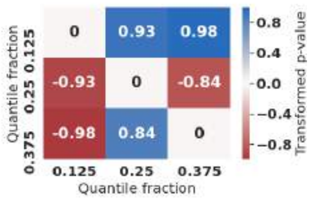
(l) Function 13



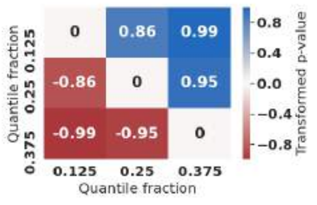
(m) Function 14



(n) Function 15



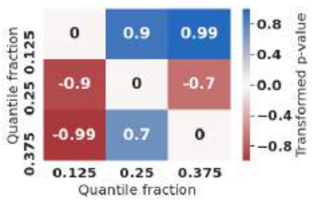
(o) Function 16



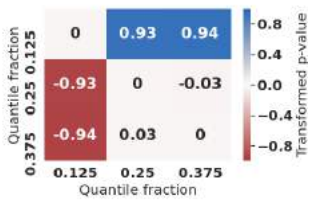
Source: Produced by the author.

Figure 49 – Comparing the best policy trained with different quantile fractions: 0.125, 0.25, 0.375. The comparisons have been made between samples of the best fitness found in the executions of the HCLPSO algorithm solving functions 17-30 from the CEC17 benchmark set. The original p-values were computed with the Wilcoxon Rank-Sum test.

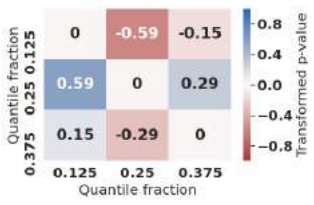
(a) Function 17



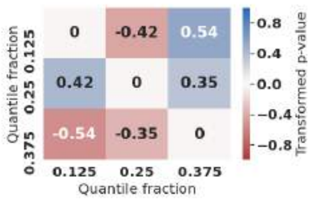
(b) Function 18



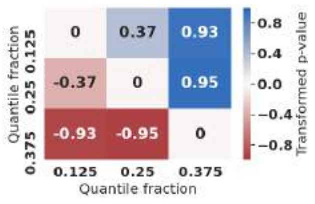
(c) Function 19



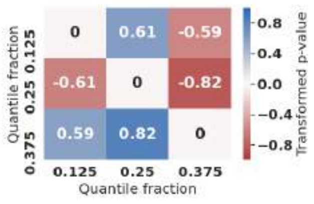
(d) Function 20



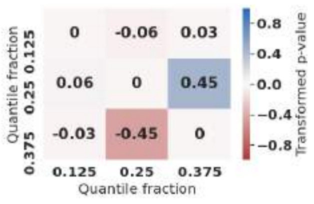
(e) Function 21



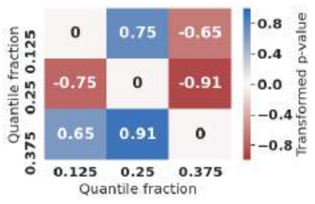
(f) Function 22



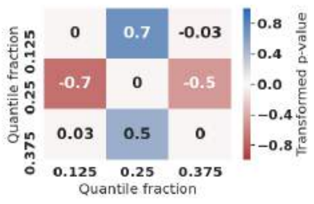
(g) Function 23



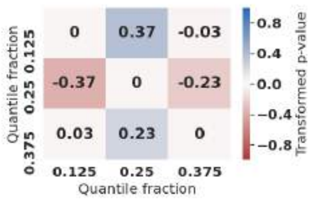
(h) Function 24



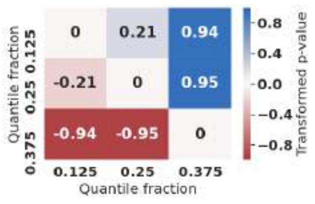
(i) Function 25



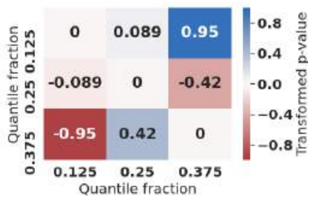
(j) Function 26



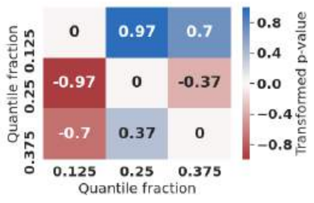
(k) Function 27



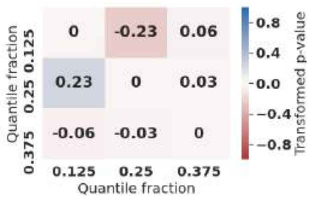
(l) Function 28



(m) Function 29



(n) Function 30



Source: Produced by the author.

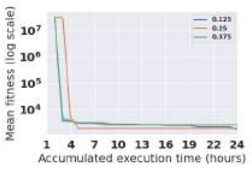
Table 17 – Mean and standard deviation (between parenthesis) of the best fitnesses found by HCLPSO with the best policy trained with different quantile fractions: 0.125, 0.25, and 0.375.

Function	**Quantile: 0.125**	**Quantile: 0.25**	**Quantile: 0.375**
1	1871.58(2570.42)	1258.53(1497.63)	1621.28(1823.51)
3	302.28(2.95)	300.72(0.74)	301.70(1.86)
4	402.69(2.21)	402.48(1.68)	402.46(2.01)
5	511.32(3.45)	512.01(4.15)	510.20(3.69)
6	600.00(0.00)	600.01(0.01)	600.01(0.00)
7	719.33(3.61)	719.74(2.91)	719.00(4.11)
8	812.42(4.42)	814.02(4.79)	811.28(3.39)
9	900.12(0.08)	900.10(0.13)	900.05(0.04)
10	1318.66(72.49)	1320.06(102.69)	1323.21(118.30)
11	1110.61(4.36)	1110.95(5.33)	1109.03(3.49)
12	7183.36(5164.44)	6848.12(3165.11)	6606.88(3718.77)
13	1654.86(143.25)	1571.99(179.09)	1743.12(295.09)
14	1451.57(16.47)	1448.30(14.26)	1451.34(15.61)
15	2340.42(932.14)	2005.76(467.43)	1890.91(319.05)
16	1626.69(39.20)	1619.82(30.65)	1631.08(46.30)
17	1756.49(17.67)	1760.80(15.94)	1768.71(32.55)
18	6672.07(2674.56)	5555.61(2253.23)	5598.84(2615.43)
19	1915.02(9.97)	1913.94(5.45)	1914.20(8.03)
20	2016.44(13.58)	2017.15(11.41)	2011.34(8.09)
21	2228.83(39.88)	2254.38(32.75)	2245.35(29.77)
22	2229.63(13.36)	2225.95(9.75)	2221.92(12.64)
23	2556.44(126.63)	2571.14(112.93)	2547.03(128.80)
24	2651.78(74.43)	2611.25(67.47)	2620.95(62.03)
25	2862.97(132.21)	2873.79(90.61)	2876.91(93.42)
26	2826.19(188.55)	2746.13(142.64)	2778.84(146.76)
27	3092.26(1.42)	3093.23(2.65)	3093.57(2.07)
28	3380.92(44.65)	3384.09(114.58)	3383.62(116.58)
29	3199.48(19.17)	3191.80(28.09)	3195.51(23.20)
30	12464.23(7271.84)	11386.59(6676.41)	12948.80(8259.95)

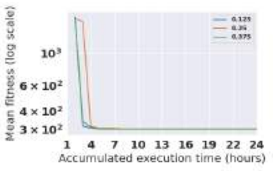
Source: Produced by the author.

Figure 50 – Mean fitness of the best policy found after some time of training in hours with quantile fractions of 0.125, 0.25, and 0.375. Each plotted point in the lines shows the mean best fitness found by HCLPSO controlled by the best policy found so far. Only functions 1-16 are shown.

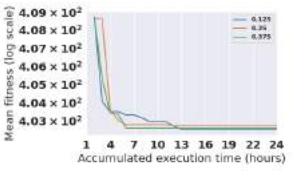
(a) Function 1



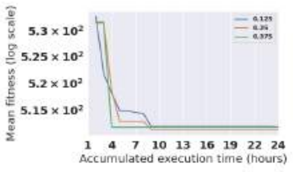
(b) Function 3



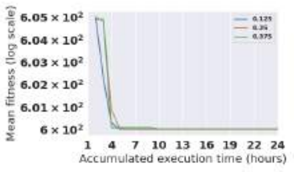
(c) Function 4



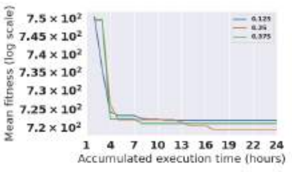
(d) Function 5



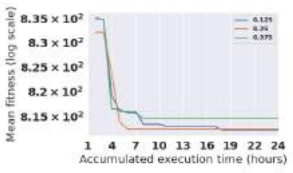
(e) Function 6



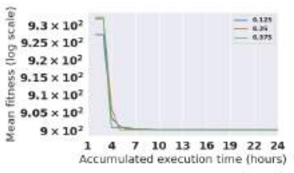
(f) Function 7



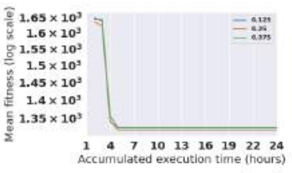
(g) Function 8



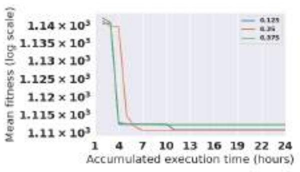
(h) Function 9



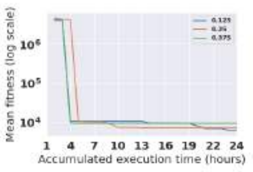
(i) Function 10



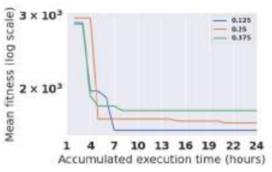
(j) Function 11



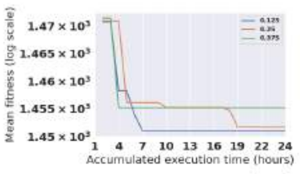
(k) Function 12



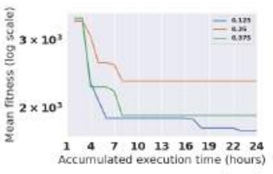
(l) Function 13



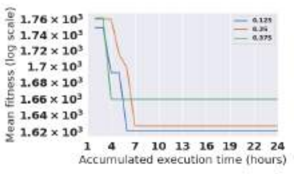
(m) Function 14



(n) Function 15



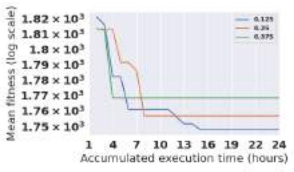
(o) Function 16



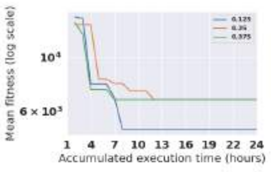
Source: Produced by the author.

Figure 51 – Mean fitness of the best policy found after some time of training in hours with quantile fractions of 0.125, 0.25, and 0.375. Each plotted point in the lines shows the mean best fitness found by HCLPSO controlled by the best policy found so far. Only functions 17-30 are shown.

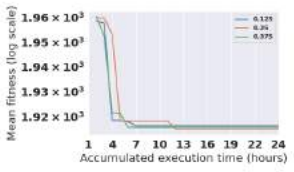
(a) Function 17



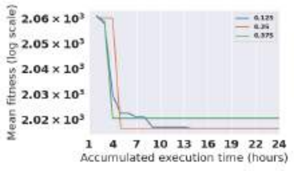
(b) Function 18



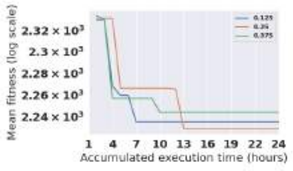
(c) Function 19



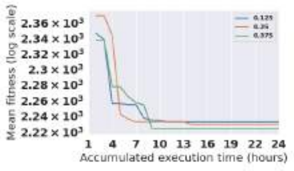
(d) Function 20



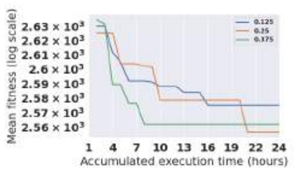
(e) Function 21



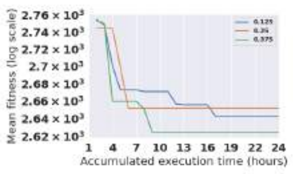
(f) Function 22



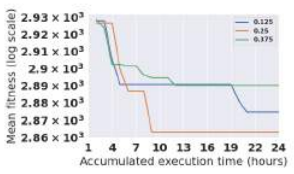
(g) Function 23



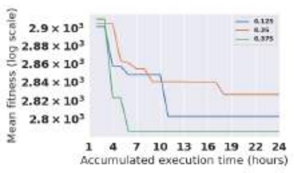
(h) Function 24



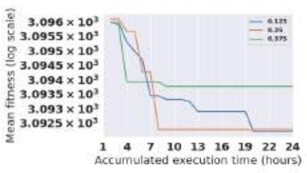
(i) Function 25



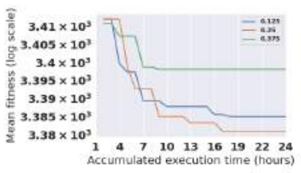
(j) Function 26



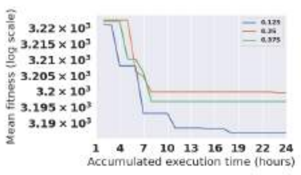
(k) Function 27



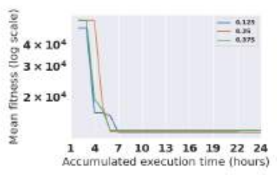
(l) Function 28



(m) Function 29



(n) Function 30

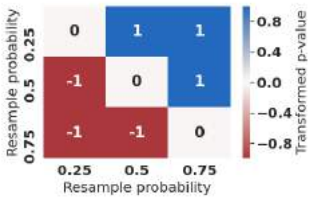


Source: Produced by the author.

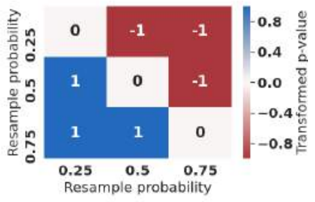
A.5 HYPERPARAMETER ANALYSIS - RESAMPLE PROBABILITY

Figure 52 – Comparing the 30% best policies trained with different resample probabilities: 0.25, 0.5, 0.75. The comparisons have been made between samples of the best fitness found in the executions of the HCLPSO algorithm solving functions 1-16 from the CEC17 benchmark set. The original p-values were computed with the Mann-Whitney U test.

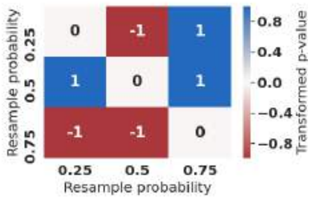
(a) Function 1



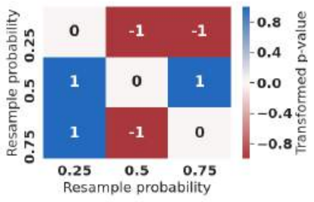
(b) Function 3



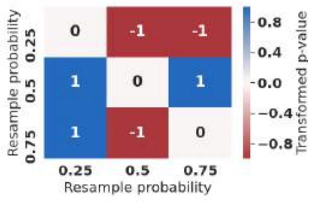
(c) Function 4



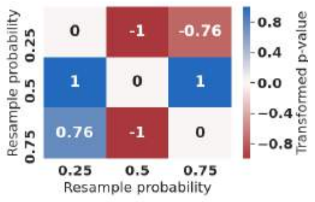
(d) Function 5



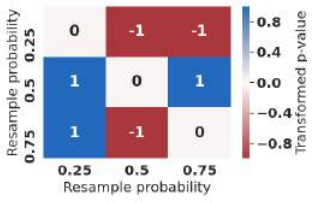
(e) Function 6



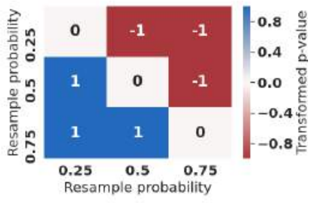
(f) Function 7



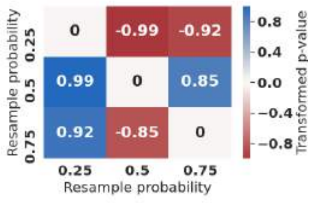
(g) Function 8



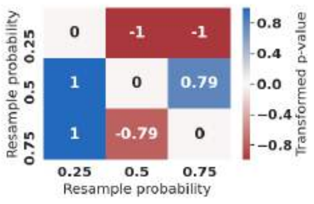
(h) Function 9



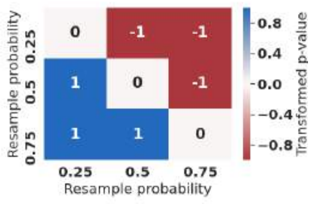
(i) Function 10



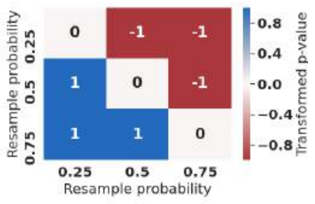
(j) Function 11



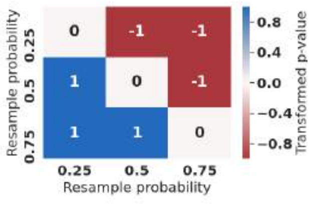
(k) Function 12



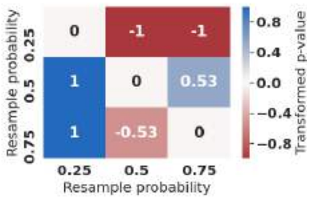
(l) Function 13



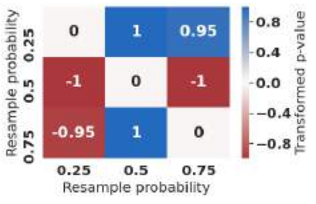
(m) Function 14



(n) Function 15



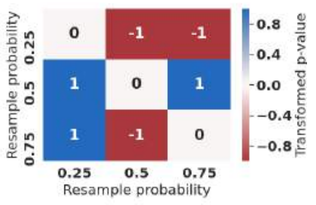
(o) Function 16



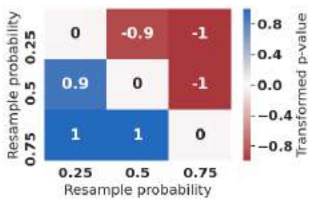
Source: Produced by the author.

Figure 53 – Comparing the 30% best policies trained with different resample probabilities: 0.25, 0.5, 0.75. The comparisons have been made between samples of the best fitness found in the executions of the HCLPSO algorithm solving functions 17-30 from the CEC17 benchmark set. The original p-values were computed with the Mann-Whitney U test.

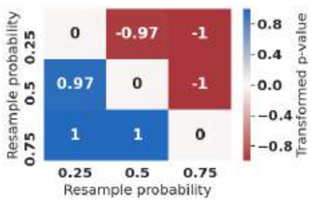
(a) Function 17



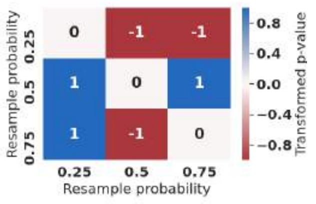
(b) Function 18



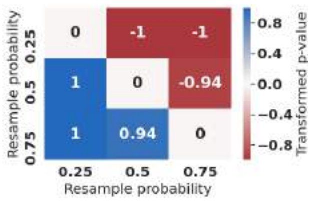
(c) Function 19



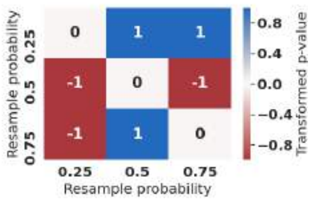
(d) Function 20



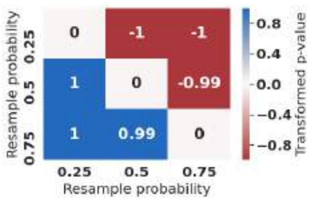
(e) Function 21



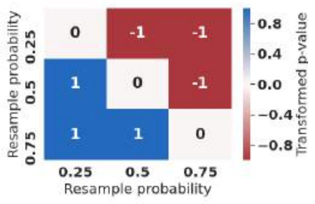
(f) Function 22



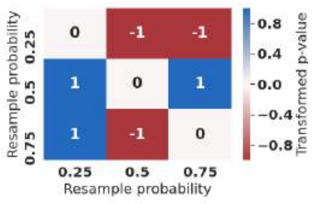
(g) Function 23



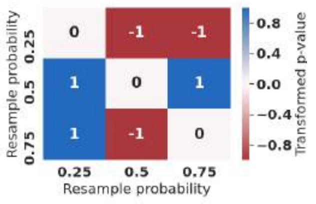
(h) Function 24



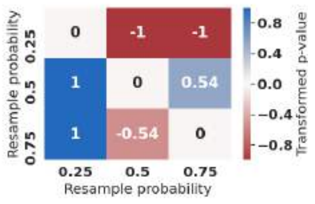
(i) Function 25



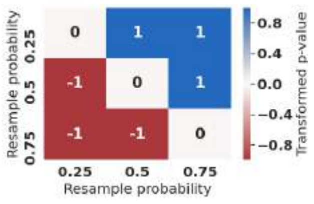
(j) Function 26



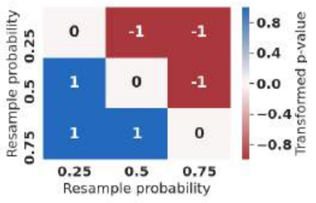
(k) Function 27



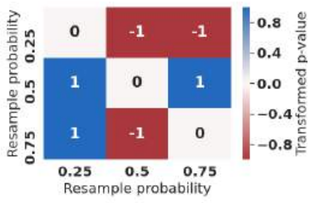
(l) Function 28



(m) Function 29



(n) Function 30



Source: Produced by the author.

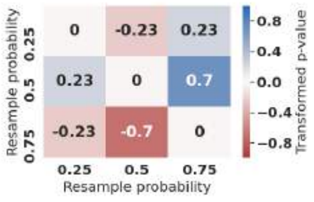
Table 18 – Mean and standard deviation (between parenthesis) of the best fitnesses found by HCLPSO with the 30% best policies trained with different resample probabilities: 0.25, 0.5, and 0.75.

Function	**Probability: 0.25**	**Probability: 0.5**	**Probability: 0.75**
1	8554.90(13973.90)	37505.78(90437.82)	51867.84(105183.66)
3	782.43(791.32)	499.53(282.06)	511.77(341.93)
4	404.35(2.09)	404.40(1.95)	404.46(1.95)
5	524.57(8.79)	523.59(8.26)	523.84(8.49)
6	602.34(3.14)	601.86(2.56)	601.95(2.48)
7	729.38(7.82)	729.04(7.62)	729.35(7.76)
8	828.06(9.84)	826.67(9.23)	827.61(9.61)
9	938.38(51.90)	933.62(47.66)	930.78(36.92)
10	1447.53(140.47)	1443.67(136.58)	1445.08(138.06)
11	1124.43(14.24)	1119.25(9.48)	1119.34(9.82)
12	379037.67(894686.73)	214801.41(539974.14)	211618.46(532502.97)
13	3138.82(1310.99)	3045.63(1203.92)	2981.10(1152.46)
14	1473.03(17.82)	1470.40(17.60)	1469.75(17.36)
15	4583.84(2502.11)	4475.72(2481.99)	4488.95(2511.58)
16	1836.32(114.72)	1840.97(114.93)	1835.11(119.48)
17	1820.76(32.15)	1816.52(30.98)	1817.83(32.85)
18	13821.35(5792.08)	13769.20(5606.61)	13659.41(5519.70)
19	1947.12(36.83)	1945.81(34.07)	1944.49(32.87)
20	2051.86(25.23)	2048.42(23.46)	2049.26(23.83)
21	2328.93(19.57)	2328.44(16.67)	2328.73(15.28)
22	2539.31(512.17)	2593.18(541.45)	2576.45(535.01)
23	2647.51(37.22)	2644.23(38.16)	2643.73(40.53)
24	2756.13(37.49)	2756.51(35.74)	2756.15(35.52)
25	2924.44(35.14)	2922.42(30.99)	2923.01(30.20)
26	2923.27(74.02)	2911.61(73.64)	2910.39(70.84)
27	3098.24(4.12)	3097.86(3.81)	3097.87(3.89)
28	3411.58(8.19)	3411.57(8.02)	3411.44(9.26)
29	3237.72(38.13)	3235.81(39.29)	3234.92(38.96)
30	27340.11(18193.29)	25022.49(15927.70)	25323.89(16111.62)

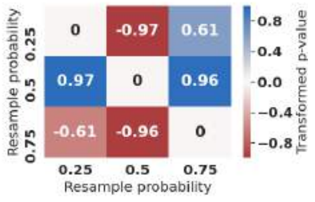
Source: Produced by the author.

Figure 54 – Comparing the best policy trained with different resample probabilities: 0.25, 0.5, and 0.75. The comparisons have been made between samples of the best fitness found in the executions of the HCLPSO algorithm solving functions 1-16 from the CEC17 benchmark set. The original p-values were computed with the Wilcoxon Rank-Sum test.

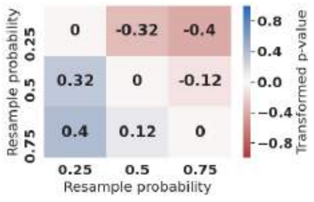
(a) Function 1



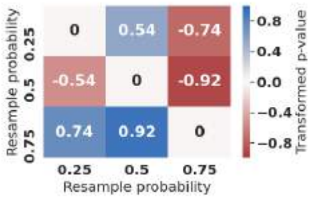
(b) Function 3



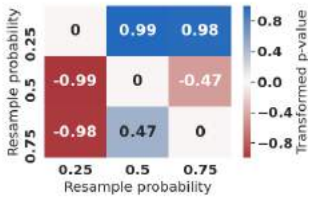
(c) Function 4



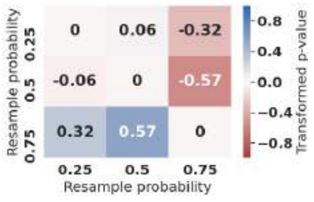
(d) Function 5



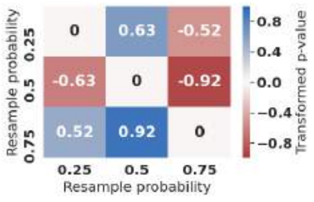
(e) Function 6



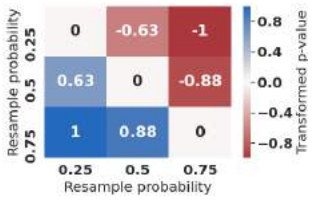
(f) Function 7



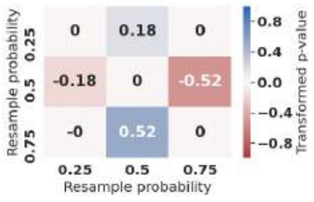
(g) Function 8



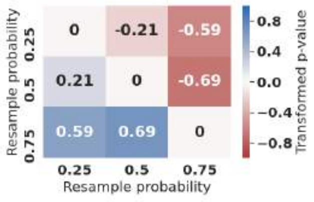
(h) Function 9



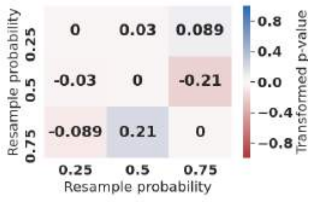
(i) Function 10



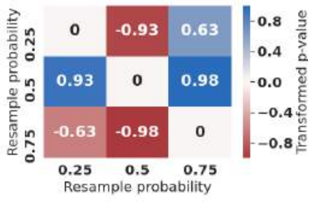
(j) Function 11



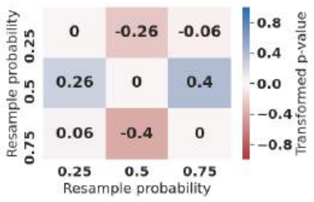
(k) Function 12



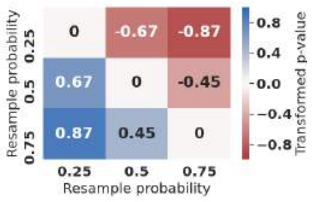
(l) Function 13



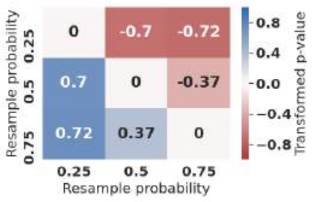
(m) Function 14



(n) Function 15



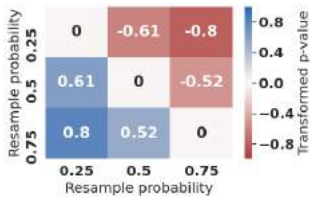
(o) Function 16



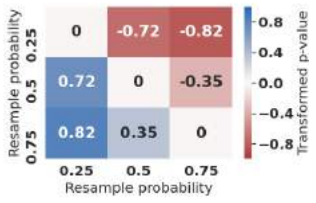
Source: Produced by the author.

Figure 55 – Comparing the best policy trained with different resample probabilities: 0.25, 0.5, and 0.75. The comparisons have been made between samples of the best fitness found in the executions of the HCLPSO algorithm solving functions 17-30 from the CEC17 benchmark set. The original p-values were computed with the Wilcoxon Rank-Sum test.

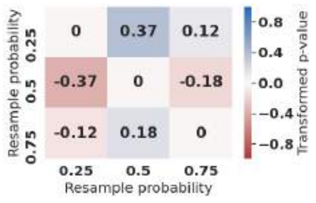
(a) Function 17

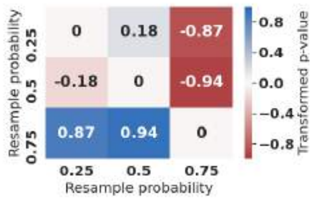


(b) Function 18

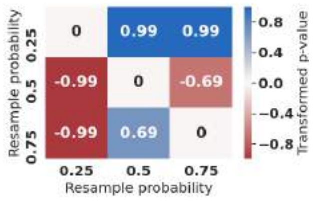


(c) Function 19

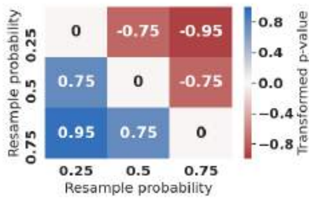




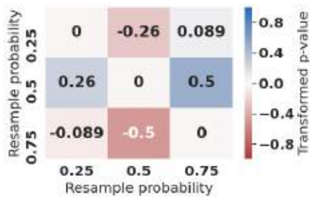
(d) Function 21



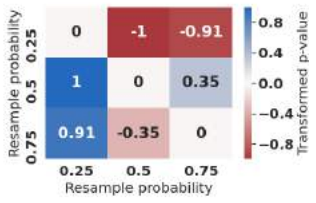
(e) Function 22



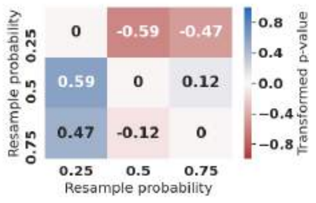
(f) Function 23



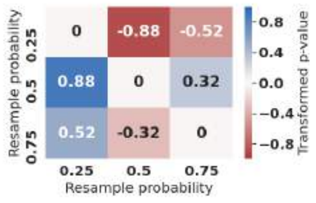
(g) Function 24



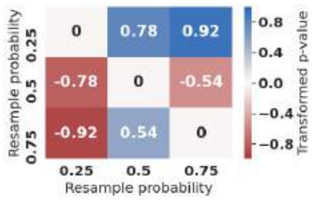
(h) Function 25



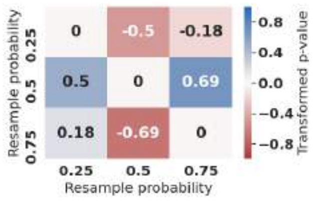
(i) Function 26



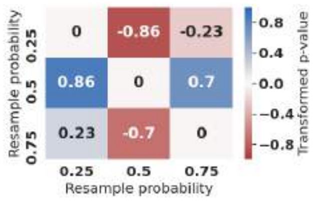
(j) Function 27



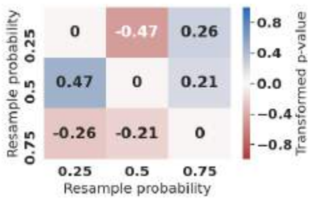
(k) Function 28



(l) Function 29



(m) Function 30



Source: Produced by the author.

Table 19 – Mean and standard deviation (between parenthesis) of the best fitnesses found by HCLPSO with the best policy trained with different resample probabilities: 0.25, 0.5, and 0.75.

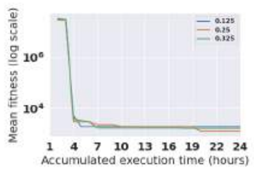
Function	**Probabilities: 0.25**	**Probabilities: 0.5**	**Probabilities: 0.75**
1	1871.58(2570.42)	1258.53(1497.63)	1621.28(1823.51)
3	302.28(2.95)	300.72(0.74)	301.70(1.86)
4	402.69(2.21)	402.48(1.68)	402.46(2.01)
5	511.32(3.45)	512.01(4.15)	510.20(3.69)
6	600.00(0.00)	600.01(0.01)	600.01(0.00)
7	719.33(3.61)	719.74(2.91)	719.00(4.11)
8	812.42(4.42)	814.02(4.79)	811.28(3.39)
9	900.12(0.08)	900.10(0.13)	900.05(0.04)
10	1318.66(72.49)	1320.06(102.69)	1323.21(118.30)
11	1110.61(4.36)	1110.95(5.33)	1109.03(3.49)
12	7183.36(5164.44)	6848.12(3165.11)	6606.88(3718.77)
13	1654.86(143.25)	1571.99(179.09)	1743.12(295.09)
14	1451.57(16.47)	1448.30(14.26)	1451.34(15.61)
15	2340.42(932.14)	2005.76(467.43)	1890.91(319.05)
16	1626.69(39.20)	1619.82(30.65)	1631.08(46.30)
17	1756.49(17.67)	1760.80(15.94)	1768.71(32.55)
18	6672.07(2674.56)	5555.61(2253.23)	5598.84(2615.43)
19	1915.02(9.97)	1913.94(5.45)	1914.20(8.03)
20	2016.44(13.58)	2017.15(11.41)	2011.34(8.09)
21	2228.83(39.88)	2254.38(32.75)	2245.35(29.77)
22	2229.63(13.36)	2225.95(9.75)	2221.92(12.64)
23	2556.44(126.63)	2571.14(112.93)	2547.03(128.80)
24	2651.78(74.43)	2611.25(67.47)	2620.95(62.03)
25	2862.97(132.21)	2873.79(90.61)	2876.91(93.42)
26	2826.19(188.55)	2746.13(142.64)	2778.84(146.76)
27	3092.26(1.42)	3093.23(2.65)	3093.57(2.07)
28	3380.92(44.65)	3384.09(114.58)	3383.62(116.58)
29	3199.48(19.17)	3191.80(28.09)	3195.51(23.20)
30	12464.23(7271.84)	11386.59(6676.41)	12948.80(8259.95)

Source: Produced by the author.

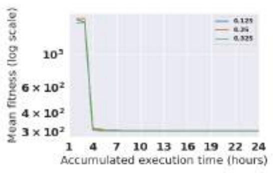
Figure 56 – Mean fitness of the best policy found after some time of training in hours with resample probabilities of 0.25, 0.5, and 0.75. Each plotted point in the lines shows the mean best fitness found by HCLPSO controlled by the best policy found so far. Only functions 1-16 are shown.

Figure 57 – Function 1

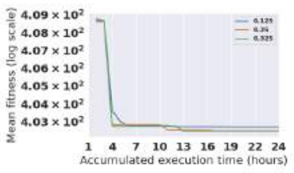
(a) Function 1



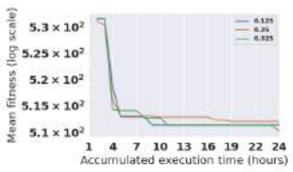
(b) Function 3



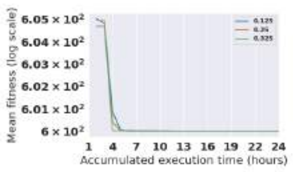
(c) Function 4



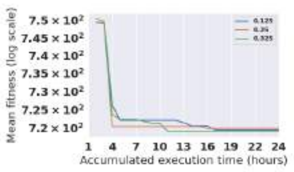
(d) Function 5



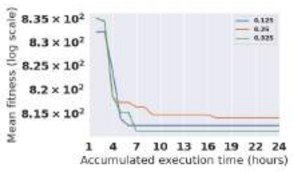
(e) Function 6



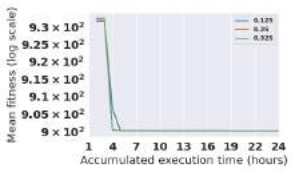
(f) Function 7



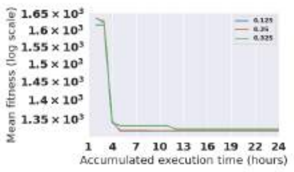
(g) Function 8



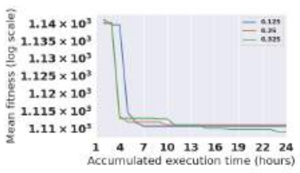
(h) Function 9



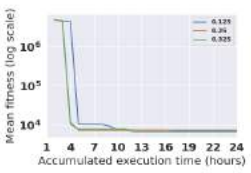
(i) Function 10



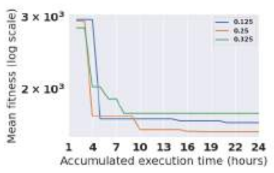
(j) Function 11



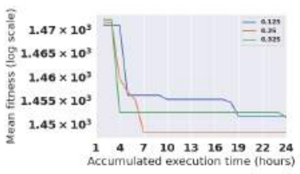
(k) Function 12



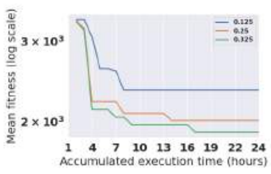
(l) Function 13



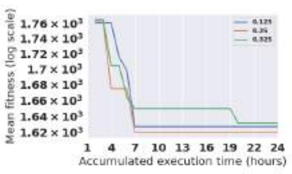
(m) Function 14



(n) Function 15



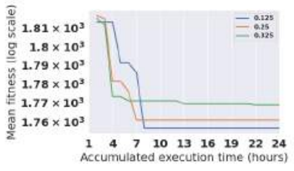
(o) Function 16



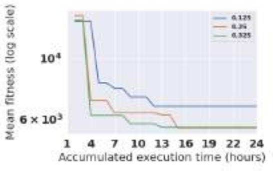
Source: Produced by the author.

Figure 58 – Mean fitness of the best policy found after some time of training in hours with resample probabilities 0.25, 0.5, and 0.75. Each plotted point in the lines shows the mean best fitness found by HCLPSO controlled by the best policy found so far. Only functions 17-30 are shown.

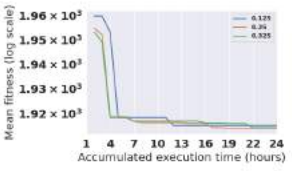
(a) Function 17



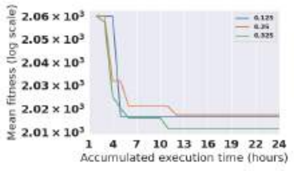
(b) Function 18



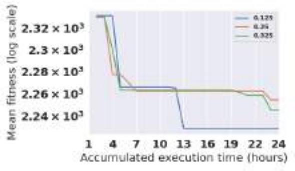
(c) Function 19



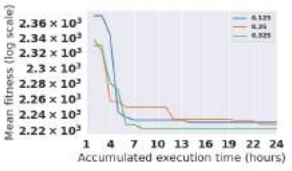
(d) Function 20



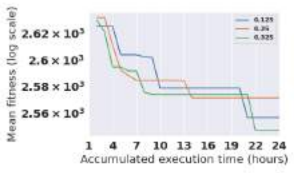
(e) Function 21



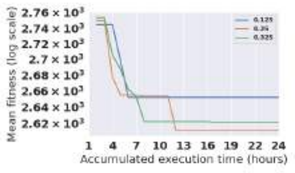
(f) Function 22



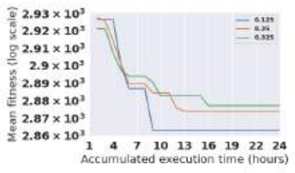
(g) Function 23



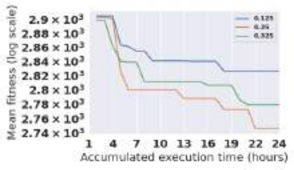
(h) Function 24



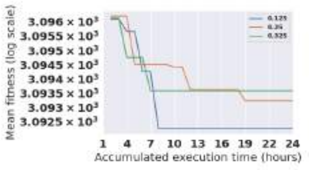
(i) Function 25



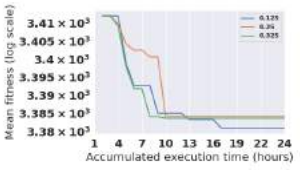
(j) Function 26



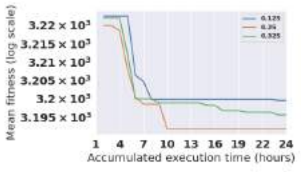
(k) Function 27



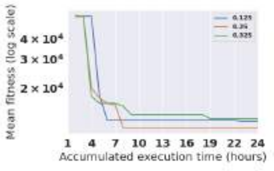
(l) Function 28



(m) Function 29



(n) Function 30

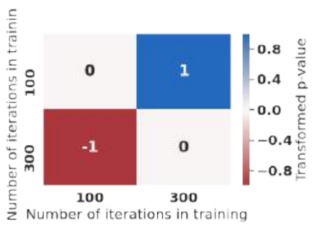


Source: Produced by the author.

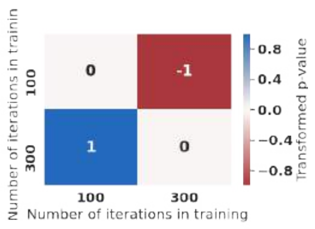
A.6 ANALYSIS OF DIFFERENT BUDGETS IN THE TRAINING AND TESTING PHASES

Figure 59 – Comparing the 30% best policies trained with 100 and 300 iterations per episode. The comparisons have been made between samples of the best fitness found in the executions of the HCLPSO algorithm solving functions 1-16 from the CEC17 benchmark set. The original p-values were computed with the Mann-Whitney U test.

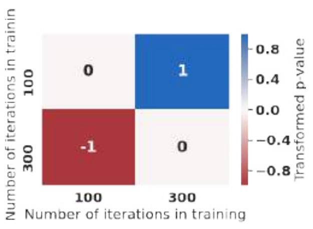
(a) Function 1



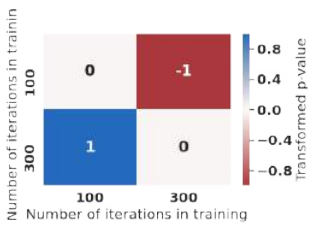
(b) Function 3



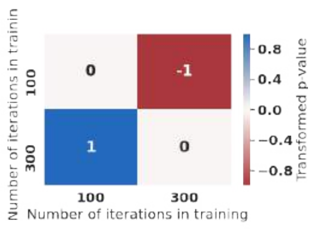
(c) Function 4



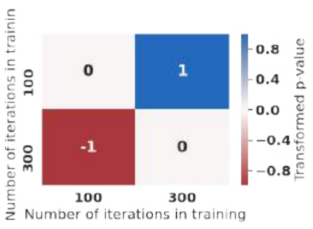
(d) Function 5



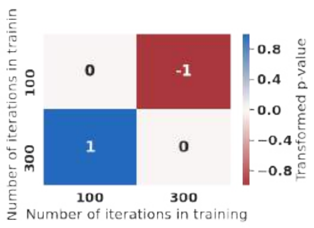
(e) Function 6



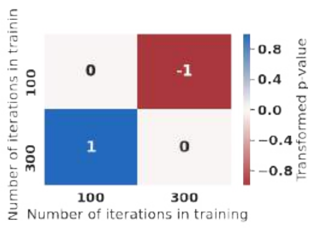
(f) Function 7



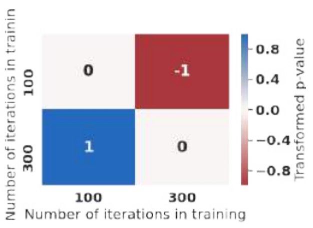
(g) Function 8



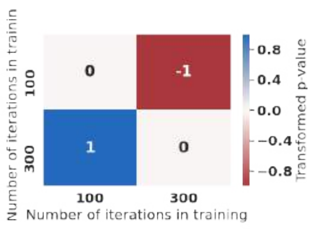
(h) Function 9



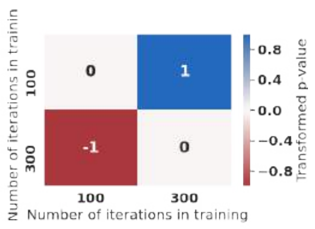
(i) Function 10



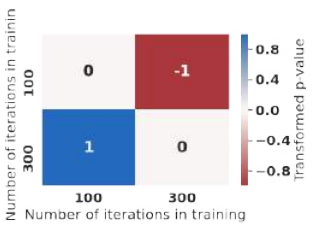
(j) Function 11



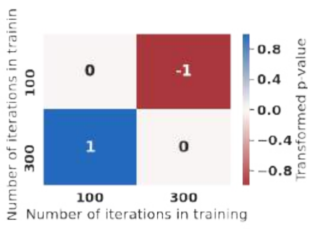
(k) Function 12



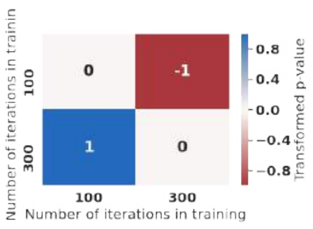
(l) Function 13



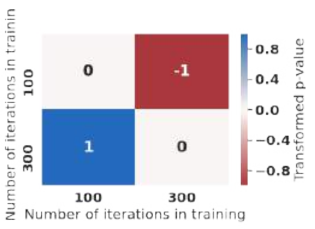
(m) Function 14



(n) Function 15



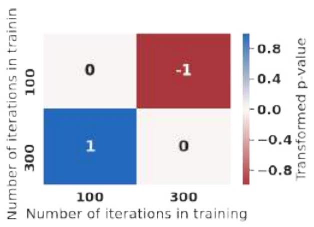
(o) Function 16



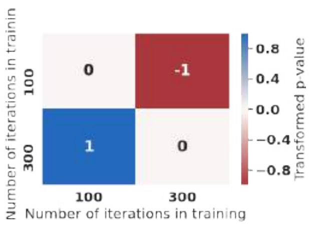
Source: Produced by the author.

Figure 60 – Comparing the 30% best policies trained 100 and 300 iterations per episode. The comparisons have been made between samples of the best fitness found in the executions of the HCLPSO algorithm solving functions 17-30 from the CEC17 benchmark set. The original p-values were computed with the Mann-Whitney U test.

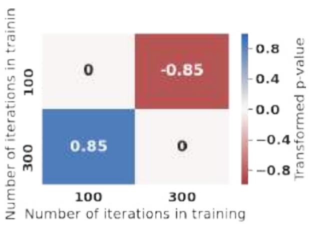
(a) Function 17



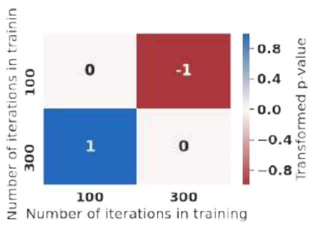
(b) Function 18



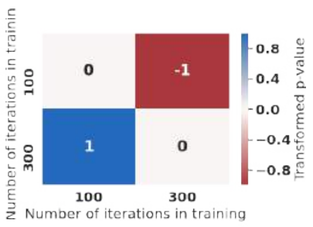
(c) Function 19



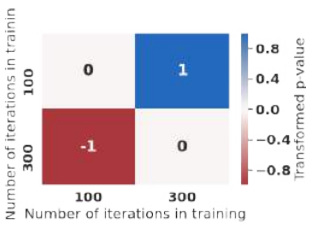
(d) Function 20



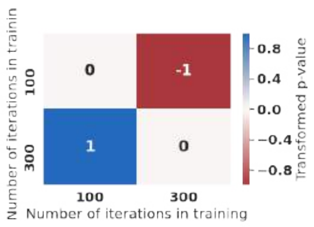
(e) Function 21



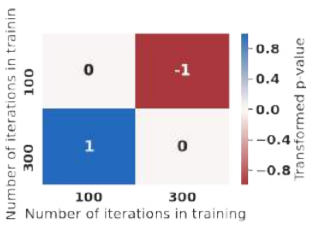
(f) Function 22



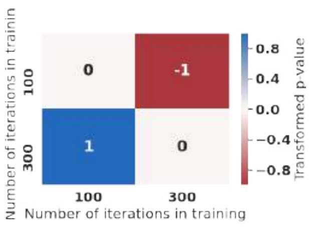
(g) Function 23



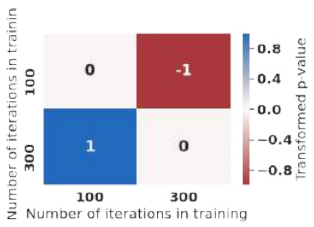
(h) Function 24



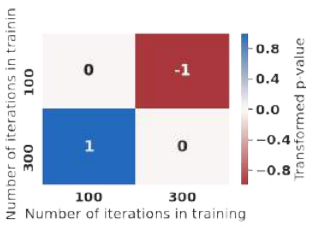
(i) Function 25



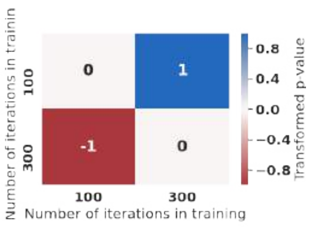
(j) Function 26



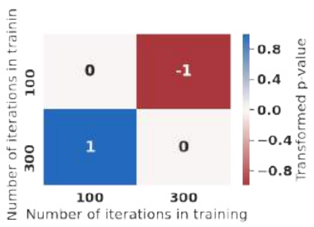
(k) Function 27



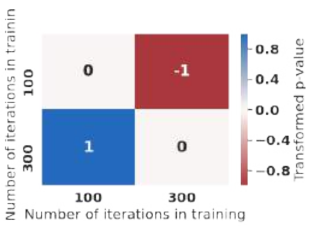
(l) Function 28



(m) Function 29



(n) Function 30



Source: Produced by the author.

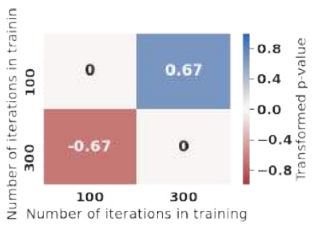
Table 20 – Mean and standard deviation (between parenthesis) of the best fitnesses found by HCLPSO with the 30% best policies trained with different 100 and 300 iterations for each episode.

Function	**300 iterations**	**100 iterations**
1	8554.90(13973.90)	92387.42(214938.81)
3	782.43(791.32)	457.68(173.54)
4	404.35(2.09)	404.65(2.05)
5	524.57(8.79)	523.46(8.08)
6	602.34(3.14)	601.84(2.56)
7	729.38(7.82)	729.65(7.79)
8	828.06(9.84)	826.29(9.02)
9	938.38(51.90)	936.58(51.15)
10	1447.53(140.47)	1456.20(144.57)
11	1124.43(14.24)	1119.96(11.07)
12	379037.67(894686.73)	308752.55(756040.51)
13	3138.82(1310.99)	3016.29(1175.49)
14	1473.03(17.82)	1470.68(18.27)
15	4583.84(2502.11)	3977.08(2162.91)
16	1836.32(114.72)	1808.43(113.16)
17	1820.76(32.15)	1810.37(27.71)
18	13821.35(5792.08)	13723.58(5707.39)
19	1947.12(36.83)	1946.41(35.34)
20	2051.86(25.23)	2048.72(23.41)
21	2328.93(19.57)	2326.97(20.90)
22	2539.31(512.17)	2487.87(470.98)
23	2647.51(37.22)	2641.24(34.38)
24	2756.13(37.49)	2753.21(42.73)
25	2924.44(35.14)	2919.72(31.99)
26	2923.27(74.02)	2912.50(79.95)
27	3098.24(4.12)	3097.32(3.41)
28	3411.58(8.19)	3411.41(9.96)
29	3237.72(38.13)	3231.66(37.12)
30	27340.11(18193.29)	24907.65(15748.79)

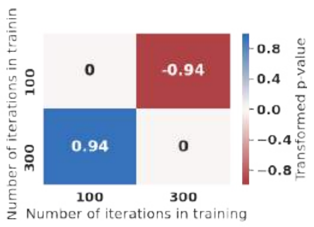
Source: Produced by the author.

Figure 61 – Comparing the best policies trained 100 and 300 iterations per episode. The comparisons have been made between samples of the best fitness found in the executions of the HCLPSO algorithm solving functions 1-16 from the CEC17 benchmark set. The original p-values were computed with the Mann-Whitney U test.

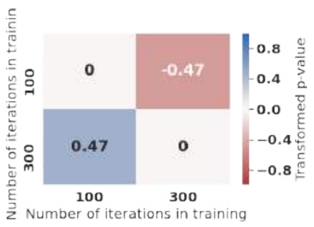
(a) Function 1



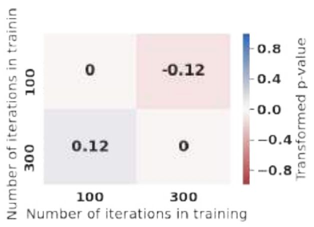
(b) Function 3



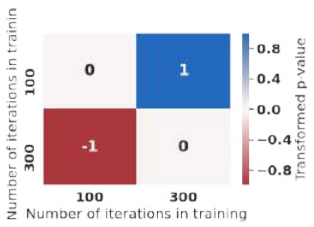
(c) Function 4



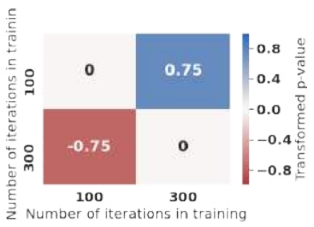
(d) Function 5



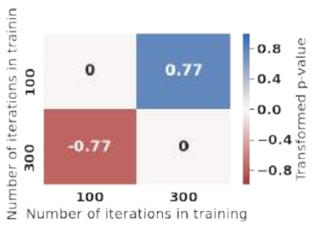
(e) Function 6



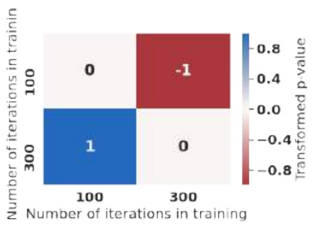
(f) Function 7



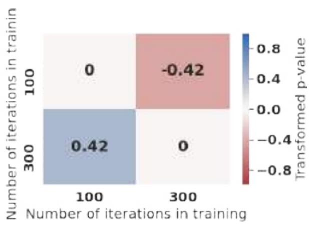
(g) Function 8



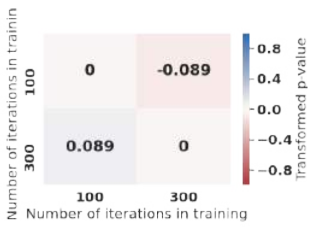
(h) Function 9



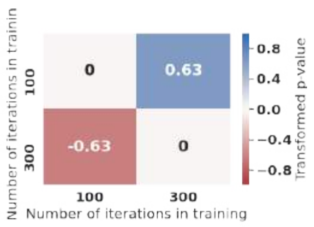
(i) Function 10



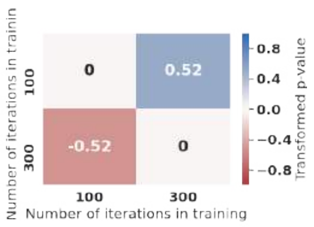
(j) Function 11



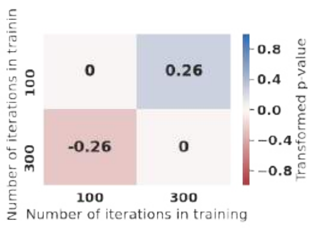
(k) Function 12



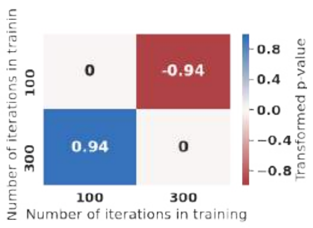
(l) Function 13



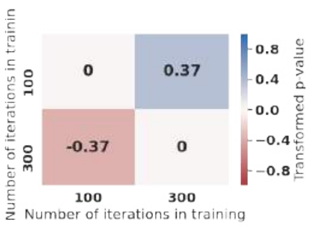
(m) Function 14



(n) Function 15



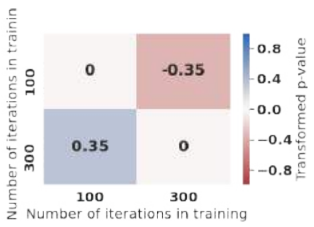
(o) Function 16



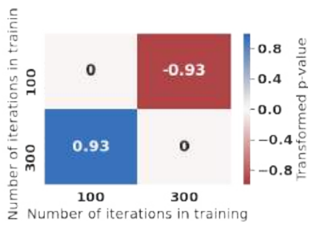
Source: Produced by the author.

Figure 62 – Comparing the best policies trained 100 and 300 iterations per episode. The comparisons have been made between samples of the best fitness found in the executions of the HCLPSO algorithm solving functions 17-30 from the CEC17 benchmark set. The original p-values were computed with the Mann-Whitney U test.

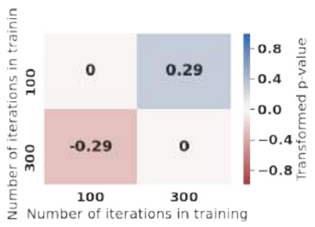
(a) Function 17



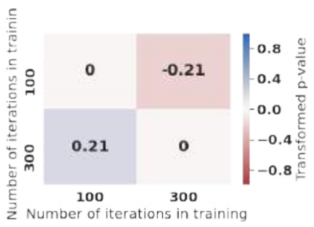
(b) Function 18



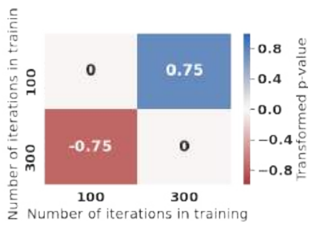
(c) Function 19



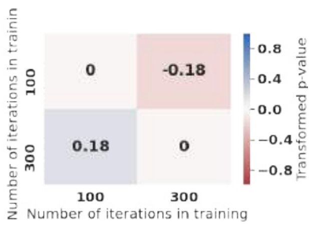
(d) Function 20



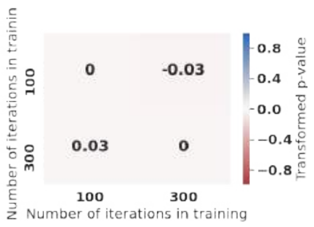
(e) Function 21



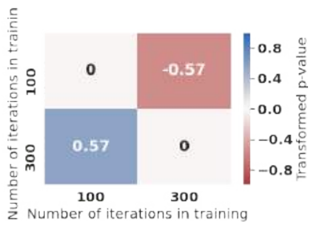
(f) Function 22



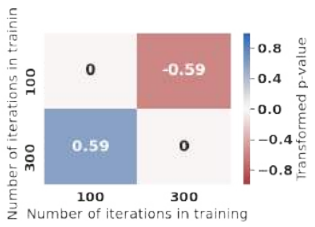
(g) Function 23



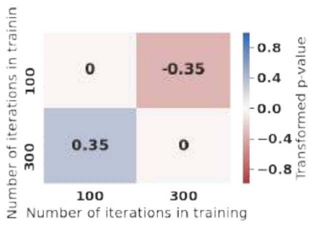
(h) Function 24



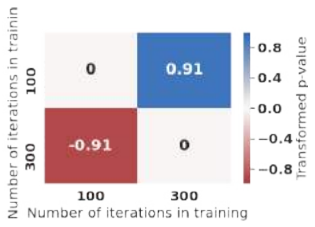
(i) Function 25



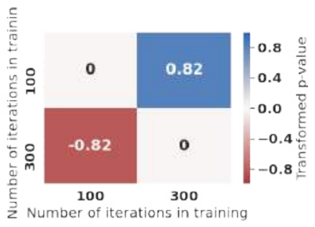
(j) Function 26



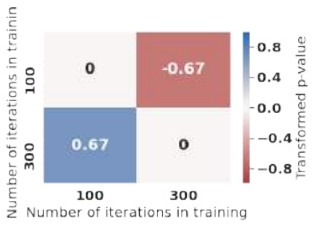
(k) Function 27



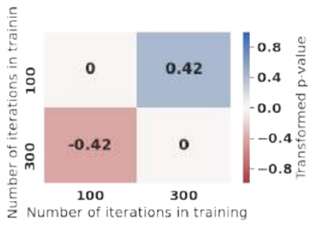
(l) Function 28



(m) Function 29

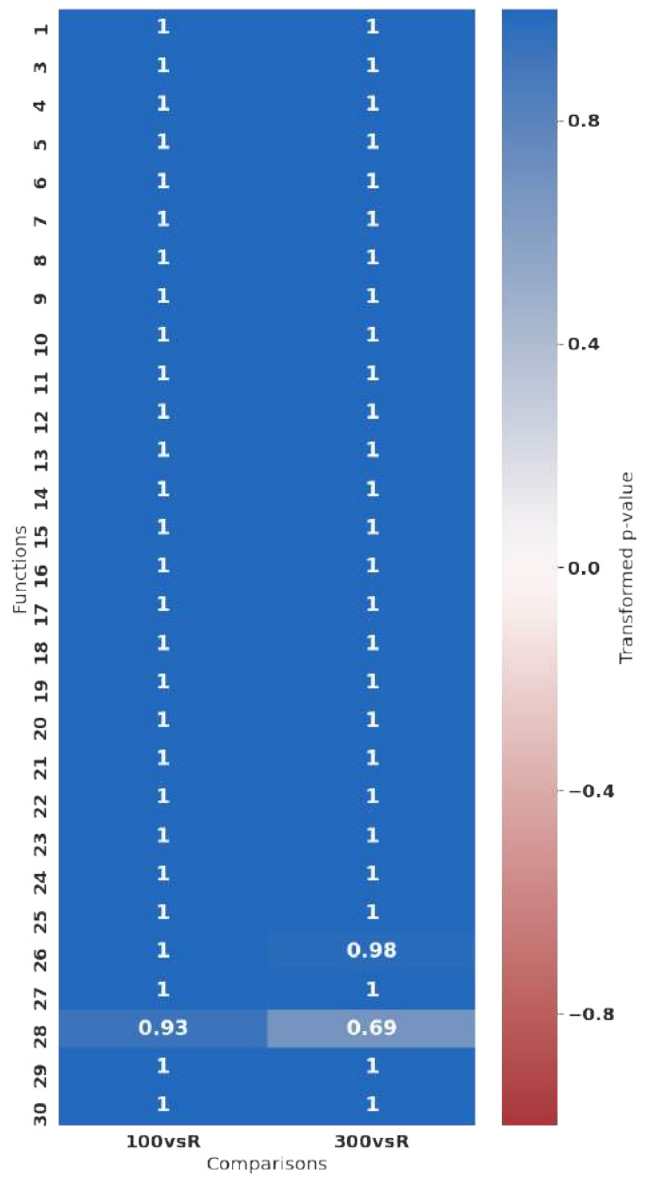


(n) Function 30



Source: Produced by the author.

Figure 63 – Transformed p-values for the comparisons between the best policy trained with 100 and 300 iterations and the random policy (100vR and 300vR, respectively).



Source: Produced by the author.

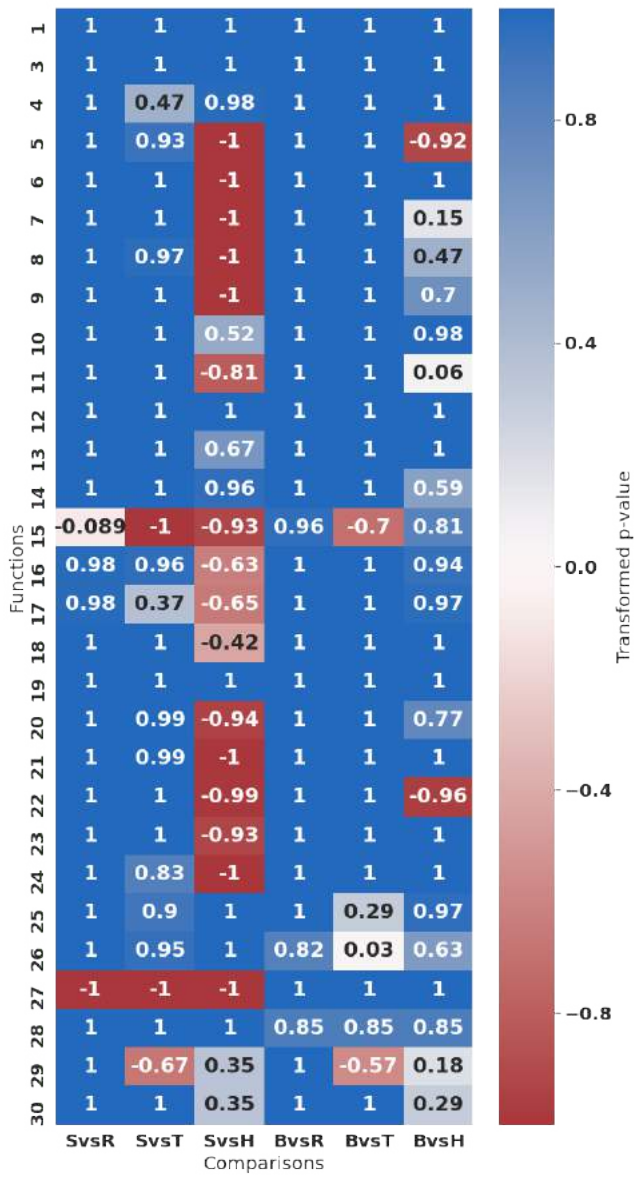
Table 21 – Mean and standard deviation (between parenthesis) of the best fitnesses found by HCLPSO with the best policies trained with different 100 and 300 iterations for each episode.

Function	**100 iterations**	**300 iterations**	**Random policy**
1	1917.12(1615.90)	1871.58(2570.42)	48289899.76(18746878.94)
3	300.74(0.59)	302.28(2.95)	2132.70(957.48)
4	402.15(1.87)	402.69(2.21)	411.58(4.62)
5	511.13(4.41)	511.32(3.45)	535.53(5.97)
6	600.03(0.04)	600.00(0.00)	605.69(2.00)
7	721.13(5.01)	719.33(3.61)	753.51(6.16)
8	813.89(4.86)	812.42(4.42)	838.64(6.31)
9	900.03(0.05)	900.12(0.08)	948.03(31.73)
10	1313.75(54.97)	1318.66(72.49)	1754.17(93.59)
11	1110.40(5.21)	1110.61(4.36)	1152.69(17.99)
12	8802.53(6387.07)	7183.36(5164.44)	7229464.29(2584268.38)
13	1814.85(501.14)	1654.86(143.25)	3212.27(1131.47)
14	1451.33(12.20)	1451.57(16.47)	1485.87(20.59)
15	1850.94(364.13)	2340.42(932.14)	5064.15(2696.46)
16	1635.25(41.91)	1626.69(39.20)	1870.38(97.67)
17	1763.17(31.37)	1756.49(17.67)	1828.83(19.02)
18	5321.73(2312.69)	6672.07(2674.56)	17893.20(4596.12)
19	1915.70(8.08)	1915.02(9.97)	1970.03(48.31)
20	2016.24(12.71)	2016.44(13.58)	2073.73(16.89)
21	2228.99(38.65)	2228.83(39.88)	2339.94(7.89)
22	2232.36(19.40)	2229.63(13.36)	2547.46(544.45)
23	2575.64(92.71)	2556.44(126.63)	2666.23(42.75)
24	2634.67(73.49)	2651.78(74.43)	2767.08(5.85)
25	2876.52(87.43)	2862.97(132.21)	2947.83(10.05)
26	2795.74(112.47)	2826.19(188.55)	2960.84(23.97)
27	3093.85(2.77)	3092.26(1.42)	3097.00(2.81)
28	3375.47(98.30)	3380.92(44.65)	3412.01(0.09)
29	3195.42(25.82)	3199.48(19.17)	3253.57(21.15)
30	12991.72(5883.13)	12464.23(7271.84)	69216.35(39271.03)

Source: Produced by the author.

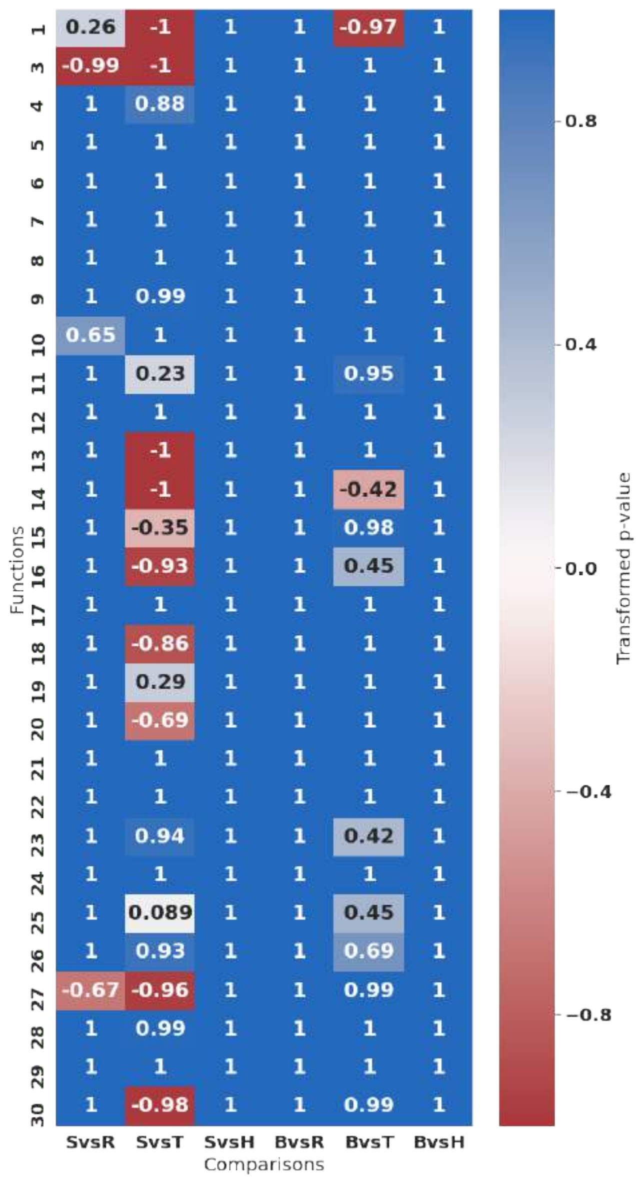
A.7 GENERALITY ASSESSMENT

Figure 64 – Transformed p-value for each of the comparisons between the selected and the best policies and the random, tuned and human-designed policies, in the experiments with HCLPSO. SvsR, SvsT, and SvsH refer to the comparisons between the selected trained policy and the random, the tuned, and the human-designed policies, respectively. BvsR, BvsT, and BvsH refer to the comparisons between the best trained policy and the random, the tuned, and the human-designed policies, respectively.



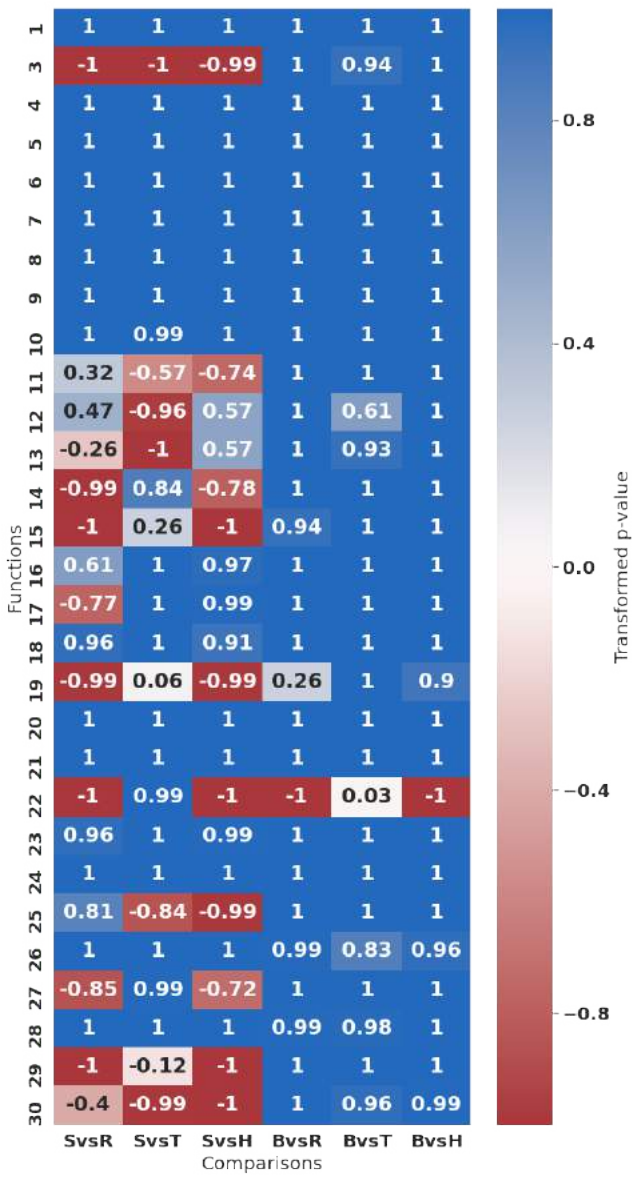
Source: Produced by the author.

Figure 65 – Transformed p-value for each of the comparisons between the selected and the best policies and the random, tuned and human-designed policies, in the experiments with DE. SvsR, SvsT, and SvsH refer to the comparisons between the selected trained policy and the random, the tuned, and the human-designed policies, respectively. BvsR, BvsT, and BvsH refer to the comparisons between the best trained policy and the random, the tuned, and the human-designed policies, respectively.



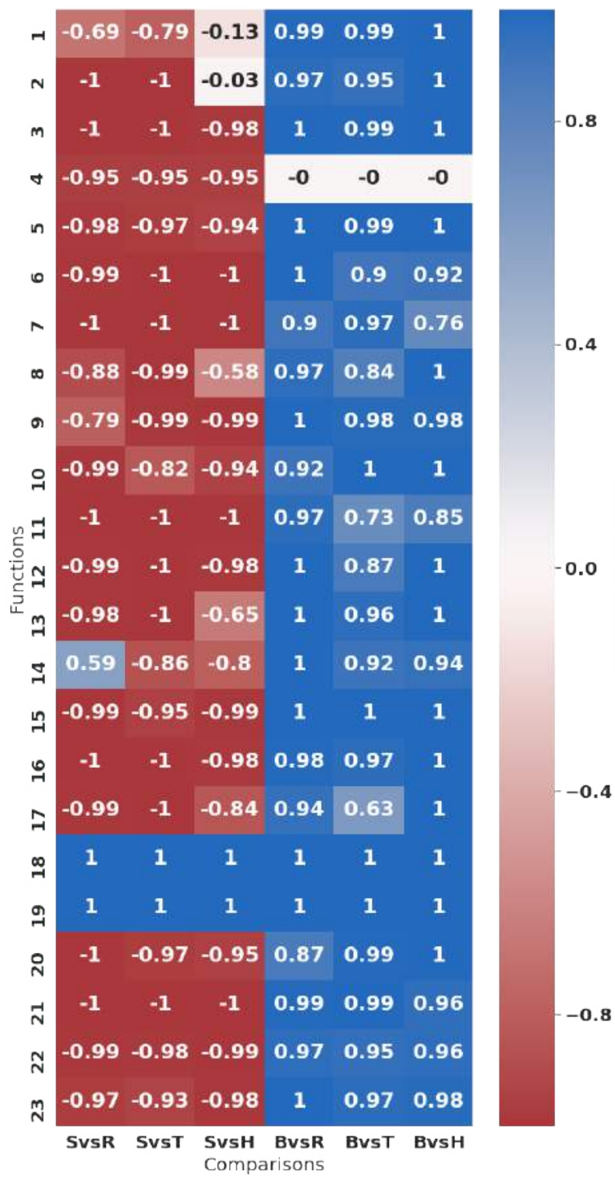
Source: Produced by the author.

Figure 66 – Transformed p-value for each of the comparisons between the selected and the best policies and the random, tuned and human-designed policies, in the experiments with FSS. SvsR, SvsT, and SvsH refer to the comparisons between the selected trained policy and the random, the tuned, and the human-designed policies, respectively. BvsR, BvsT, and BvsH refer to the comparisons between the best trained policy and the random, the tuned, and the human-designed policies, respectively.



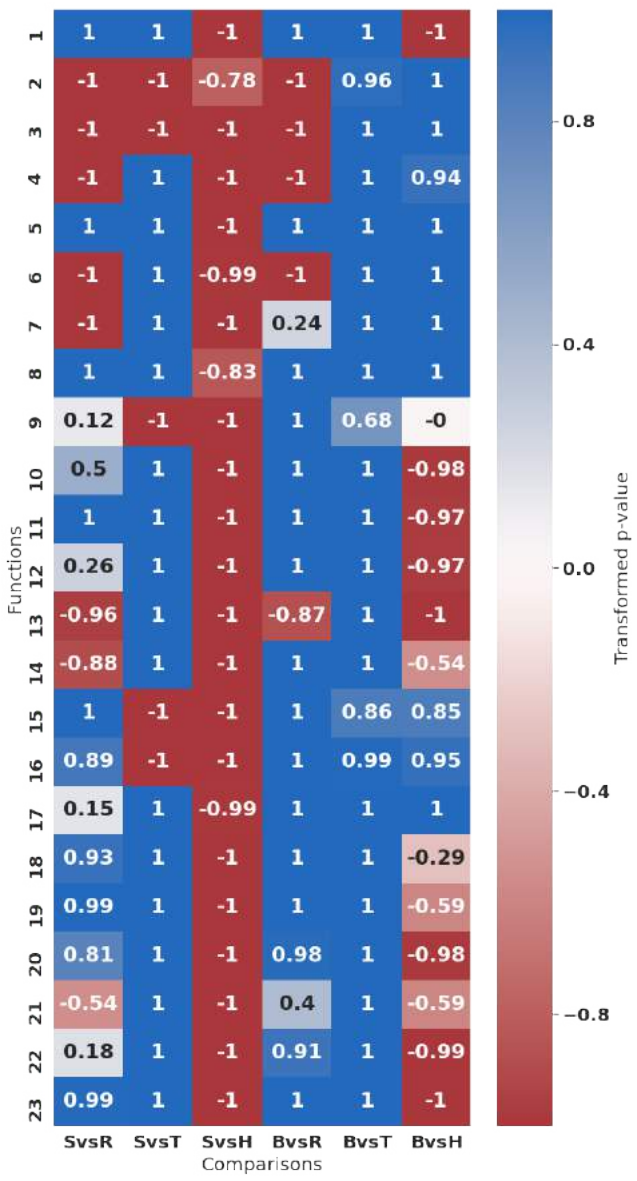
Source: Produced by the author.

Figure 67 – Transformed p-value for each of the comparisons between the selected and the best policies and the random, tuned and human-designed policies, in the experiments with binary GA. SvsR, SvsT, and SvsH refer to the comparisons between the selected trained policy and the random, the tuned, and the human-designed policies, respectively. BvsR, BvsT, and BvsH refer to the comparisons between the best trained policy and the random, the tuned, and the human-designed policies, respectively.



Source: Produced by the author.

Figure 68 – Transformed p-value for each of the comparisons between the selected and the best policies and the random, tuned and human-designed policies, in the experiments with ACO. SvsR, SvsT, and SvsH refer to the comparisons between the selected trained policy and the random, the tuned, and the human-designed policies, respectively. BvsR, BvsT, and BvsH refer to the comparisons between the best trained policy and the random, the tuned, and the human-designed policies, respectively.



Source: Produced by the author.

Table 22 – Mean of the best fitnesses found by HCLPSO with its parameters controlled by the selected policies, the best policies, a human-designed policie, a random policy, and the same algorithm with static parameters defined by I/F-Race.

Functions	**Selected**	**Best**	**Tuned**	**Human-designed**	**Random**
1	5718.30	3057.93	39784.05	14624.41	53631491.47
3	305.00	302.52	309.15	946.75	2268.20
4	403.49	402.59	405.68	406.50	411.15
5	519.35	512.17	523.24	510.81	537.28
6	600.40	600.00	601.21	600.09	605.47
7	725.26	720.58	742.52	721.90	754.20
8	822.31	813.37	823.66	812.18	839.21
9	901.95	900.12	2070.04	900.24	957.77
10	1417.93	1370.14	1705.89	1471.25	1709.24
11	1116.57	1110.55	1134.14	1112.18	1144.02
12	68489.16	5304.08	1654627.59	1524043.92	5660436.91
13	1895.52	1689.17	9468.94	2147.14	3451.58
14	1451.33	1450.04	1491.09	1456.16	1482.77
15	4822.70	2293.16	2197.94	3766.02	5193.20
16	1779.90	1634.52	1854.31	1681.37	1842.91
17	1811.01	1760.89	1799.03	1772.13	1837.47
18	7652.16	6124.17	18887.83	10407.79	19042.83
19	1914.37	1916.03	2063.66	1930.34	1966.40
20	2027.66	2020.26	2043.78	2020.16	2089.52
21	2322.04	2229.43	2332.20	2312.54	2339.59
22	2290.09	2230.56	3117.18	2228.58	2394.23
23	2624.64	2555.47	2637.35	2606.78	2669.70
24	2757.57	2629.10	2767.82	2741.20	2766.18
25	2910.95	2853.58	2923.68	2932.04	2951.10
26	2853.43	2758.13	2940.31	2901.24	2953.84
27	3104.48	3092.89	3097.63	3097.60	3096.87
28	3411.82	3386.63	3411.82	3411.82	3412.01
29	3208.87	3202.04	3194.94	3194.91	3246.37
30	12557.70	14555.50	39278.70	18085.06	62427.46

Source: Produced by the author.

Table 23 – Mean of the best fitnesses found by DE with its parameters controlled by the selected policies, the best policies, a human-designed policie, a random policy, and the same algorithm with static parameters defined by I/F-Race.

Functions	**Selected**	**Best**	**Tuned**	**Human-designed**	**Random**
1	489.00	100.00	100.00	6019784427.70	534348270.78
3	814.90	300.00	300.00	124195.78	300.00
4	401.80	400.00	401.42	673.15	405.74
5	517.03	506.31	534.09	584.82	593.03
6	600.04	600.00	600.06	641.02	639.70
7	722.39	716.32	730.18	877.39	806.21
8	813.15	807.47	845.73	886.43	893.18
9	900.01	900.00	1305.77	2389.08	2343.36
10	1346.06	1178.83	1547.55	2647.20	1578.83
11	1103.06	1100.87	1103.36	3035.59	1173.19
12	1531.93	1277.21	2642.18	126355151.18	5771.95
13	1314.28	1303.18	1305.20	2899814.47	1664.62
14	1422.35	1403.05	1404.25	1721.08	1490.24
15	1502.11	1500.68	1502.55	38258.77	1558.45
16	1619.14	1606.05	1650.71	2139.70	2051.25
17	1750.47	1716.03	1835.49	1879.79	2015.80
18	1824.90	1812.57	1821.80	6036138.87	1915.44
19	1902.63	1900.60	1902.38	16831.58	1940.01
20	2030.84	2000.25	2025.84	2087.87	2207.73
21	2307.27	2293.28	2343.50	2375.58	2373.56
22	2561.05	2305.56	3426.52	4120.49	3444.63
23	2615.00	2585.60	2636.55	2776.34	3029.69
24	2746.55	2751.61	2765.06	2856.52	2876.24
25	2905.28	2896.78	2936.33	3297.36	2990.99
26	2925.51	2828.98	2901.48	3898.70	3621.44
27	3161.57	3074.53	3112.22	3200.00	3118.38
28	3278.00	3226.70	3273.88	3299.76	3278.66
29	3141.06	3126.79	3237.00	3656.63	3385.81
30	3230.32	3201.85	3202.79	41442.71	3530.41

Source: Produced by the author.

Table 24 – Mean of the best fitnesses found by FSS with its parameters controlled by the selected policies, the best policies, a human-designed policie, a random policy, and the same algorithm with static parameters defined by I/F-Race.

Functions	**Selected**	**Best**	**Tuned**	**Human-designed**	**Random**
1	14942026.17	1203173.75	75552469.22	51456296.37	36571310.56
3	39476.46	14310.30	15456.95	30034.36	20294.10
4	408.05	404.12	411.91	410.04	412.91
5	514.47	507.91	523.21	538.52	537.11
6	602.86	601.00	612.93	613.06	610.96
7	740.14	730.49	872.12	767.74	754.38
8	811.50	807.72	910.04	843.88	834.92
9	905.97	900.80	2037.81	1012.01	1018.68
10	1592.47	1397.19	1771.96	2142.61	2253.92
11	1273.43	1201.45	1309.96	1245.91	1306.06
12	3776875.12	1312499.11	1948267.91	5800850.15	5171175.16
13	17177.42	8395.02	9754.04	21732.65	13163.57
14	1570.77	1507.15	1585.87	1552.03	1539.79
15	12887.50	2220.71	12847.41	5733.02	4735.37
16	1847.78	1636.85	2057.68	1912.50	1822.27
17	1909.62	1785.02	2015.94	1936.62	1842.45
18	19538.06	13370.53	35615.87	25453.00	29238.39
19	3042.66	2413.03	3122.74	2413.44	2637.25
20	2093.35	2062.18	2199.16	2160.95	2131.13
21	2313.33	2313.02	2334.95	2339.86	2334.72
22	3353.83	3719.91	3766.32	2396.82	2320.42
23	2600.64	2528.26	3731.70	2591.91	2647.01
24	2740.62	2735.15	2772.11	2766.33	2765.20
25	2939.25	2908.22	2927.10	2916.55	2944.85
26	2914.97	2905.47	2919.47	2930.56	2958.69
27	3110.55	3091.02	3118.46	3095.26	3096.24
28	3376.07	3357.80	3387.53	3391.42	3386.46
29	3306.35	3195.95	3317.77	3285.69	3259.44
30	279463.28	59619.66	113083.90	127455.36	288677.55

Source: Produced by the author.

Table 25 – Mean of the best fitnesses found by binary GA with its parameters controlled by the selected policies, the best policies, a human-designed policy, a random policy, and the same algorithm with static parameters defined by I/F-Race.

Functions	**Selected**	**Best**	**Tuned**	**Human-designed**	**Random**
1	8034.80	8709.10	8367.20	8213.80	8485.10
3	19721.00	21304.30	20916.30	20077.40	20499.75
4	14472.95	15950.05	15744.85	14865.45	15094.45
5	8945.55	8946.00	8946.00	8945.85	8946.00
6	10042.25	10116.25	10097.00	10073.25	10105.75
7	24472.85	24486.95	24479.25	24484.90	24480.55
8	16552.50	16768.35	16619.40	16634.35	16667.55
9	2250.90	2271.50	2268.50	2258.00	2262.50
10	3417.20	3486.90	3458.60	3460.95	3447.55
11	13258.50	13405.05	13363.20	13327.80	13394.55
12	13915.60	14031.05	14012.25	13972.05	13992.45
13	5344.25	5426.20	5437.70	5361.30	5393.20
14	1250.75	1738.60	1736.10	1461.45	1640.20
15	27095.40	28231.50	27769.90	27539.30	27514.35
16	4614.35	8927.00	7229.60	6821.55	8371.75
17	9223.50	9255.30	9251.70	9243.45	9248.70
18	5555.40	5556.90	5556.60	5556.15	5556.75
19	0.00	126.15	0.00	0.00	0.00
20	0.00	0.00	0.00	0.00	0.00
21	14222.55	14575.05	14350.10	14304.30	14501.65
22	8887.20	9118.80	9056.40	9054.05	9096.55
23	16284.55	16703.15	16457.55	16468.25	16683.65

Source: Produced by the author.

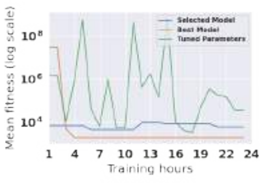
Table 26 – Mean of the best fitnesses found by ACO with its parameters controlled by the selected policies, the best policies, a human-designed policy, a random policy, and the same algorithm with static parameters defined by I/F-Race.

Functions	**Selected**	**Best**	**Tuned**	**Human-designed**	**Random**
1	0.00047	0.00047	0.00000	0.00047	0.00036
3	0.00033	0.00034	0.00034	0.00034	0.00035
4	0.00033	0.00035	0.00035	0.00035	0.00040
5	0.00034	0.00035	0.00000	0.00035	0.00036
6	0.00038	0.00040	0.00000	0.00039	0.00037
7	0.00365	0.00376	0.00000	0.00369	0.00413
8	0.00376	0.00398	0.00000	0.00389	0.00397
9	0.00490	0.00513	0.00000	0.00491	0.00337
10	0.00372	0.00409	0.00409	0.00404	0.00371
11	0.00017	0.00019	0.00000	0.00020	0.00017
12	0.00017	0.00017	0.00000	0.00018	0.00015
13	0.00016	0.00017	0.00000	0.00018	0.00016
14	0.00016	0.00017	0.00000	0.00019	0.00018
15	0.00016	0.00017	0.00000	0.00018	0.00016
16	0.00173	0.00195	0.00200	0.00193	0.00156
17	0.00165	0.00198	0.00172	0.00185	0.00161
18	0.00162	0.00181	0.00000	0.00171	0.00151
19	0.00162	0.00185	0.00121	0.00189	0.00144
20	0.00176	0.00195	0.00143	0.00191	0.00155
21	0.00010	0.00010	0.00000	0.00011	0.00010
22	0.00010	0.00010	0.00000	0.00011	0.00010
23	0.00010	0.00011	0.00000	0.00011	0.00009

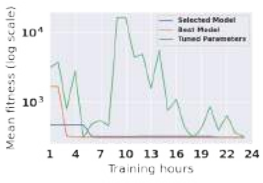
Source: Produced by the author.

Figure 69 – Mean fitness found by HCLPSO with the selected policies, the best policies, and static tuned parameters, after some time of training in hours. Only functions 1-16 are shown.

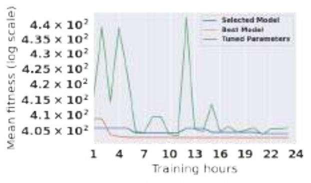
(a) Function 1



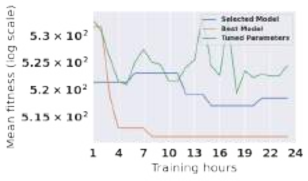
(b) Function 3



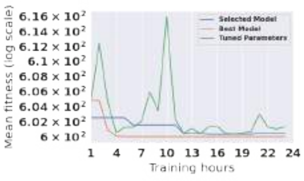
(c) Function 4



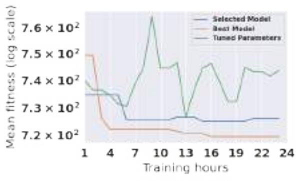
(d) Function 5



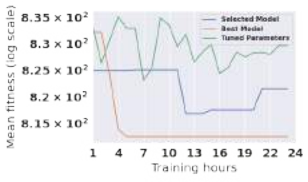
(e) Function 6



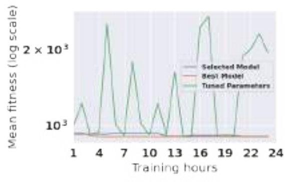
(f) Function 7



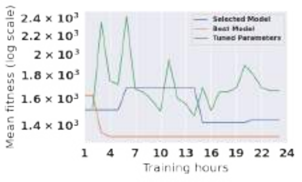
(g) Function 8



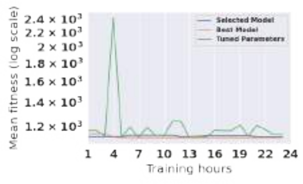
(h) Function 9



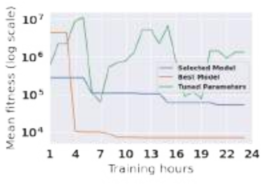
(i) Function 10



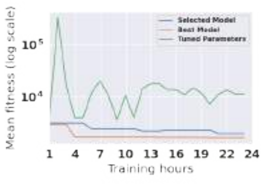
(j) Function 11



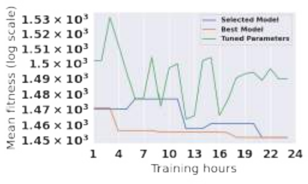
(k) Function 12



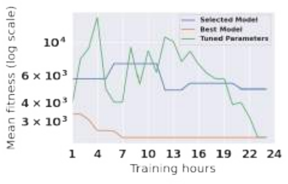
(l) Function 13



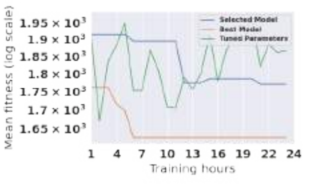
(m) Function 14



(n) Function 15



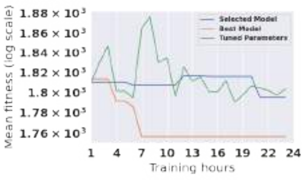
(o) Function 16



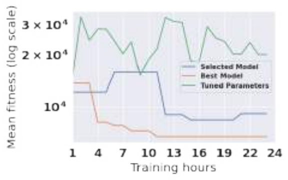
Source: Produced by the author.

Figure 70 – Mean fitness found by HCLPSO with the selected policies, the best policies, and static tuned parameters, after some time of training in hours. Only functions 17-30 are shown.

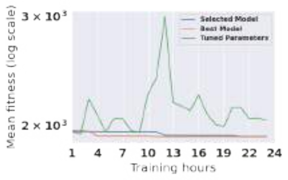
(a) Function 17



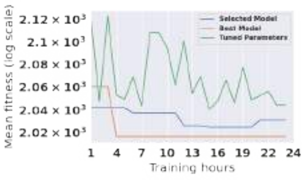
(b) Function 18



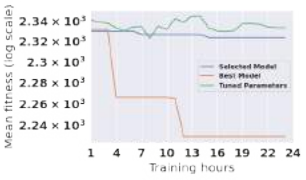
(c) Function 19



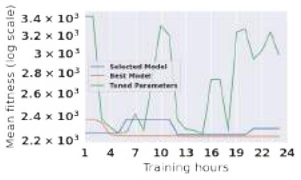
(d) Function 20



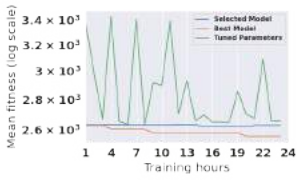
(e) Function 21



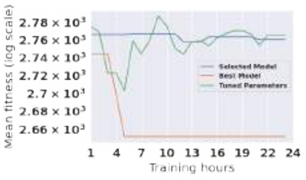
(f) Function 22



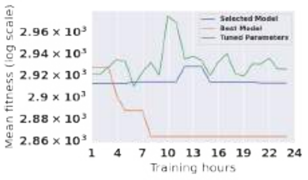
(g) Function 23



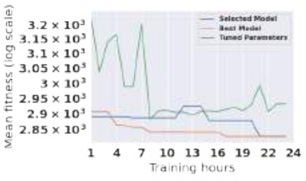
(h) Function 24



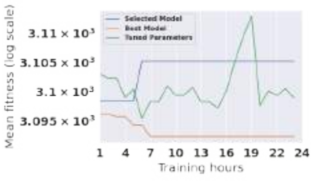
(i) Function 25



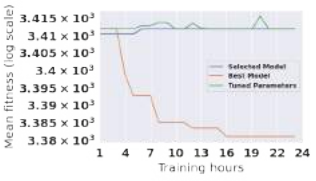
(j) Function 26



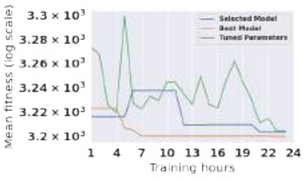
(k) Function 27



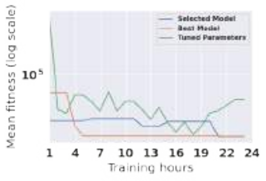
(l) Function 28



(m) Function 29



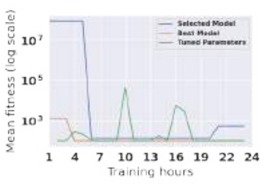
(n) Function 30



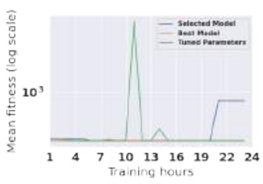
Source: Produced by the author.

Figure 71 – Mean fitness found by DE with the selected policies, the best policies, and static tuned parameters, after some time of training in hours. Only functions 1-16 are shown.

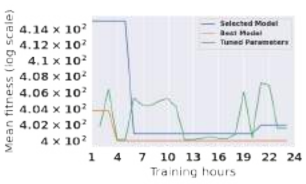
(a) Function 1



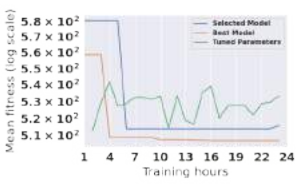
(b) Function 3



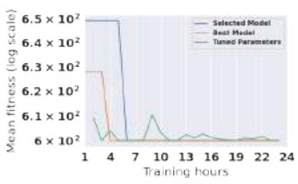
(c) Function 4



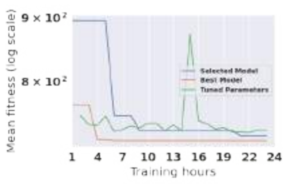
(d) Function 5



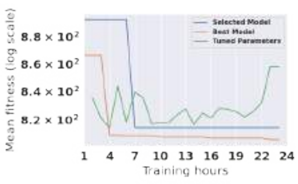
(e) Function 6



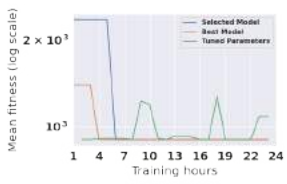
(f) Function 7



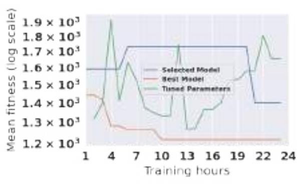
(g) Function 8



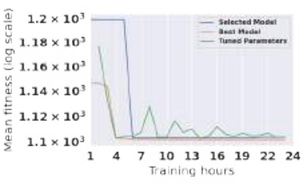
(h) Function 9



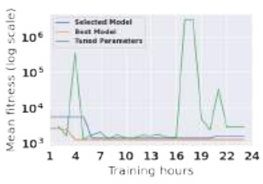
(i) Function 10



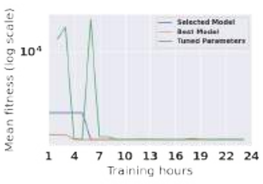
(j) Function 11



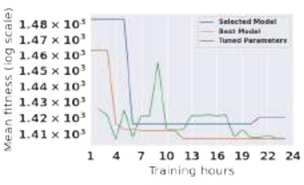
(k) Function 12



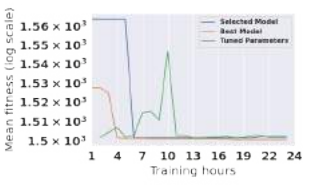
(l) Function 13



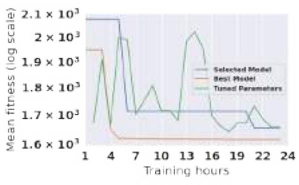
(m) Function 14



(n) Function 15



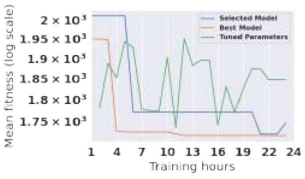
(o) Function 16



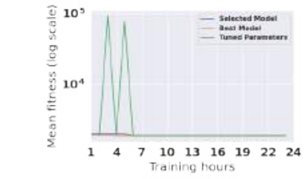
Source: Produced by the author.

Figure 72 – Mean fitness found by DE with the selected policies, the best policies, and static tuned parameters, after some time of training in hours. Only functions 17-30 are shown.

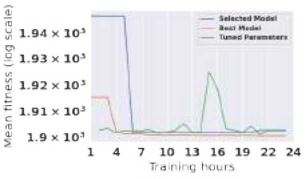
(a) Function 17



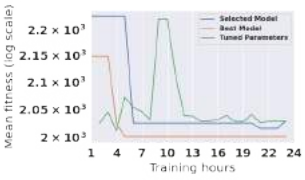
(b) Function 18



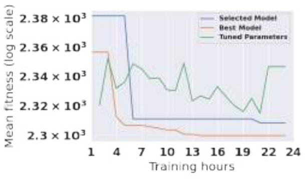
(c) Function 19



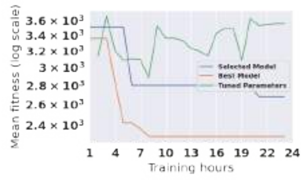
(d) Function 20



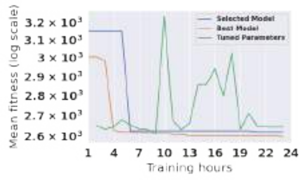
(e) Function 21



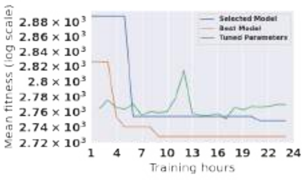
(f) Function 22



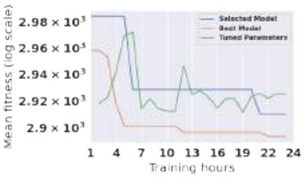
(g) Function 23



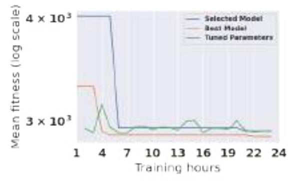
(h) Function 24



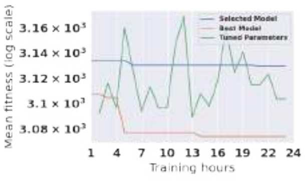
(i) Function 25



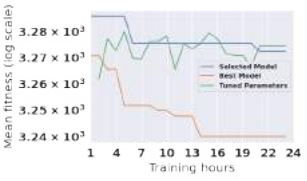
(j) Function 26



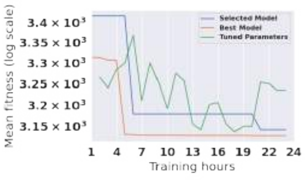
(k) Function 27



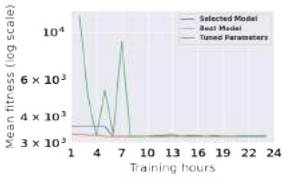
(l) Function 28



(m) Function 29



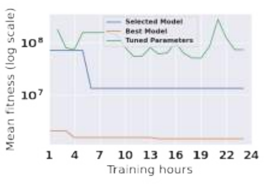
(n) Function 30



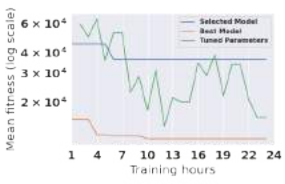
Source: Produced by the author.

Figure 73 – Mean fitness found by FSS with the selected policies, the best policies, and static tuned parameters, after some time of training in hours. Only functions 1-16 are shown.

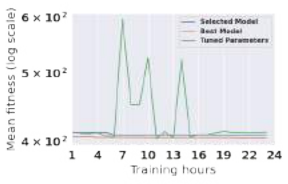
(a) Function 1



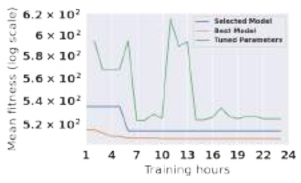
(b) Function 3



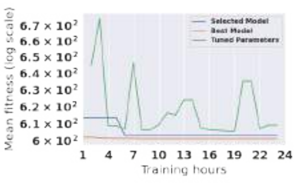
(c) Function 4



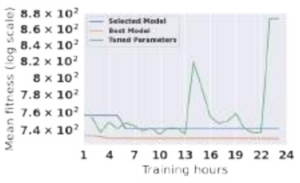
(d) Function 5



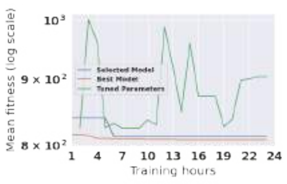
(e) Function 6



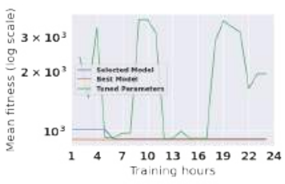
(f) Function 7



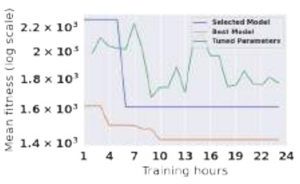
(g) Function 8



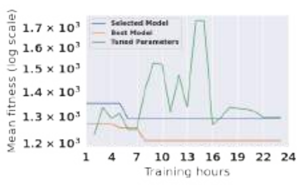
(h) Function 9



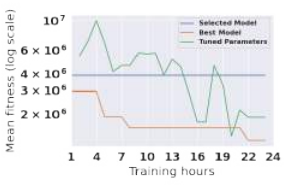
(i) Function 10



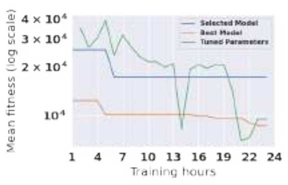
(j) Function 11



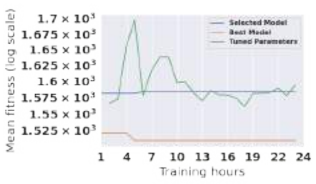
(k) Function 12



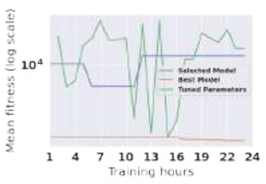
(l) Function 13



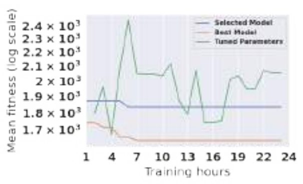
(m) Function 14



(n) Function 15



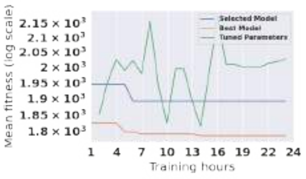
(o) Function 16



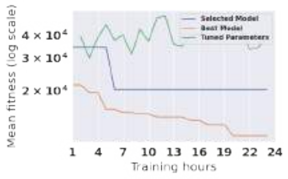
Source: Produced by the author.

Figure 74 – Mean fitness found by FSS with the selected policies, the best policies, and static tuned parameters, after some time of training in hours. Only functions 17-30 are shown.

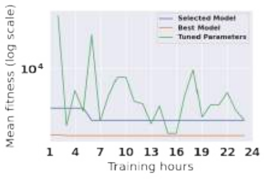
(a) Function 17



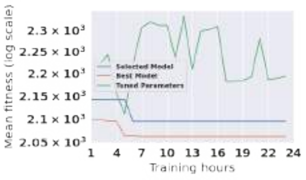
(b) Function 18



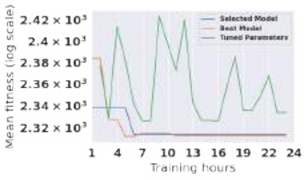
(c) Function 19



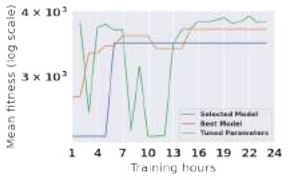
(d) Function 20



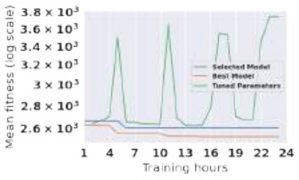
(e) Function 21



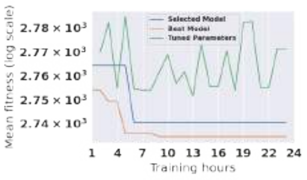
(f) Function 22



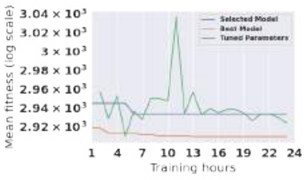
(g) Function 23



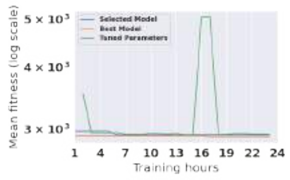
(h) Function 24



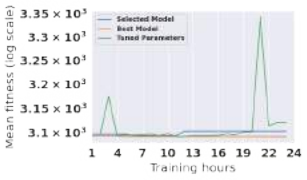
(i) Function 25



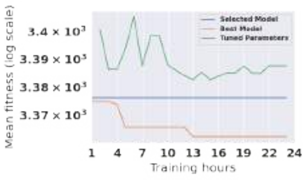
(j) Function 26



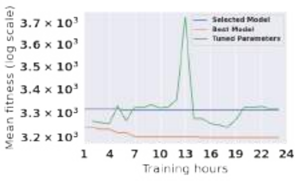
(k) Function 27



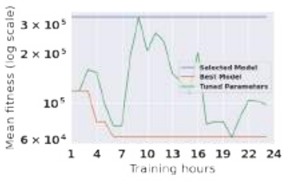
(l) Function 28



(m) Function 29



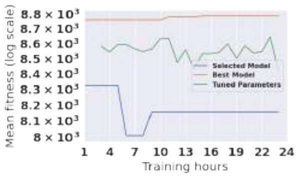
(n) Function 30



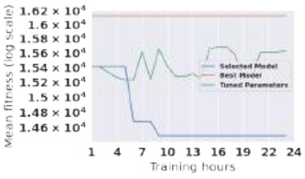
Source: Produced by the author.

Figure 75 – Mean fitness found by binary GA with the selected policies, the best policies, and static tuned parameters, after some time of training in hours. Only problem instances 1-12 are shown.

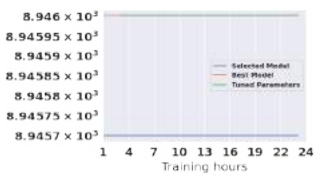
(a) Function 1



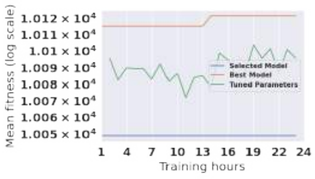
(b) Function 3



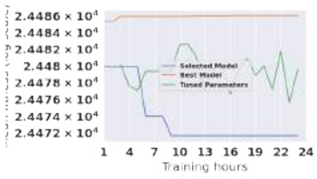
(c) Function 4



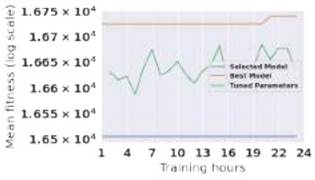
(d) Function 5



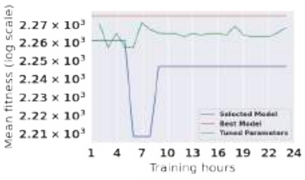
(e) Function 6



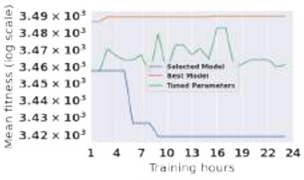
(f) Function 7



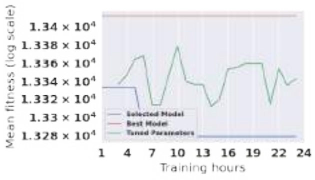
(g) Function 8



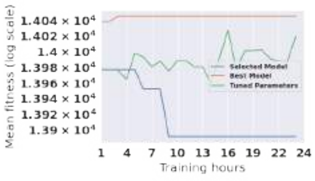
(h) Function 9



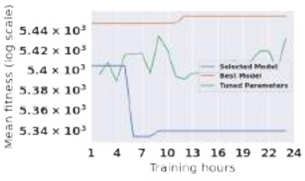
(i) Function 10



(j) Function 11



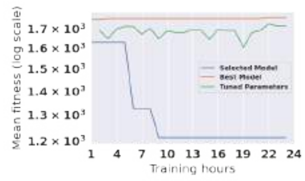
(k) Function 12



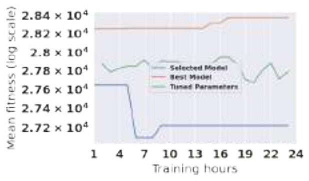
Source: Produced by the author.

Figure 76 – Mean fitness found by binary GA with the selected policies, the best policies, and static tuned parameters, after some time of training in hours. Only functions 13-23 are shown.

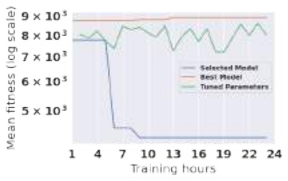
(a) Function 13



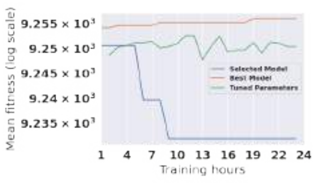
(b) Function 14



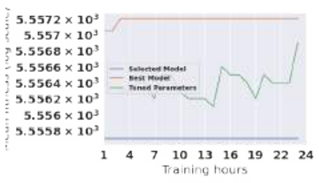
(c) Function 15



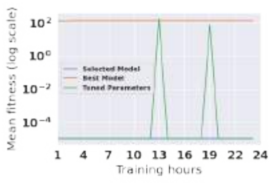
(d) Function 16



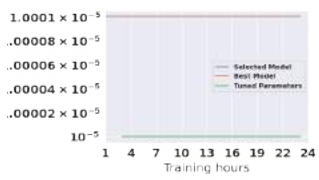
(e) Function 17



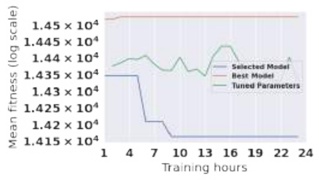
(f) Function 18



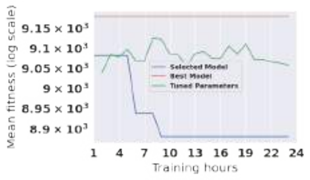
(g) Function 19



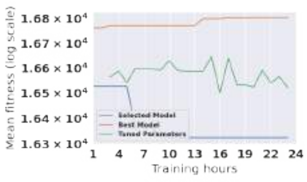
(h) Function 20



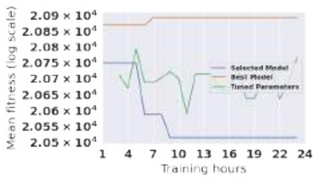
(i) Function 21



(j) Function 22



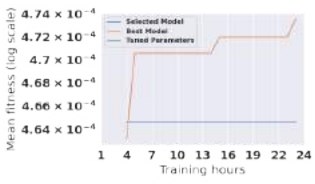
(k) Function 23



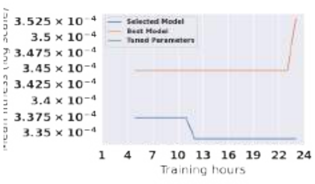
Source: Produced by the author.

Figure 77 – Mean fitness found by ACO with the selected policies, the best policies, and static tuned parameters, after some time of training in hours. Only problem instances 1-12 are shown.

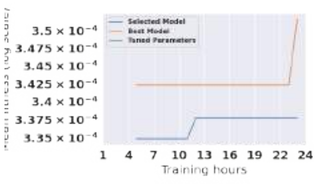
(a) Function 1



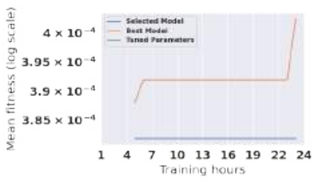
(b) Function 3



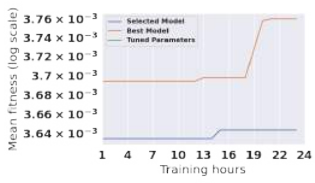
(c) Function 4



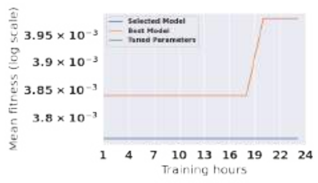
(d) Function 5



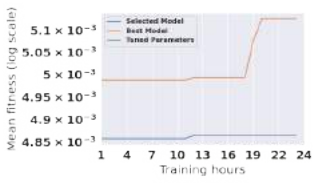
(e) Function 6



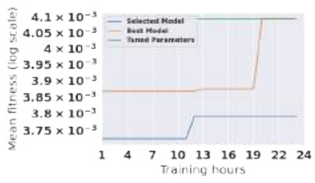
(f) Function 7



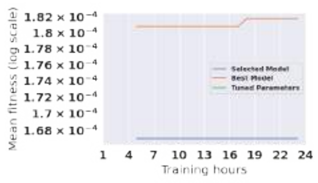
(g) Function 8



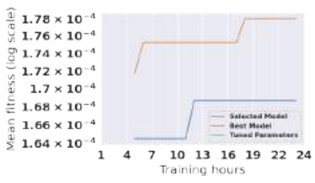
(h) Function 9



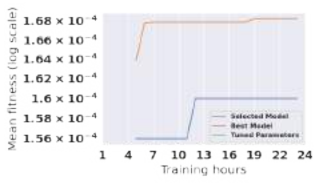
(i) Function 10



(j) Function 11



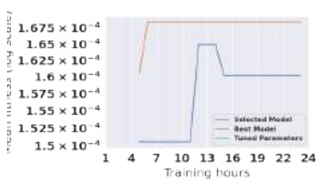
(k) Function 12



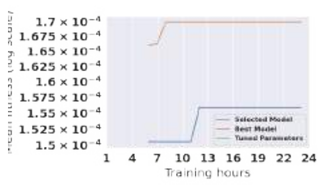
Source: Produced by the author.

Figure 78 – Mean fitness found by ACO with the selected policies, the best policies, and static tuned parameters, after some time of training in hours. Only functions 13-23 are shown.

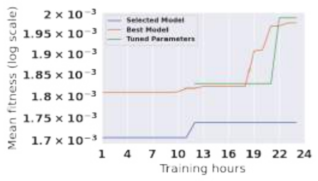
(a) Function 13



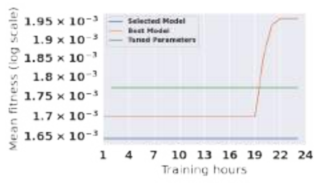
(b) Function 14



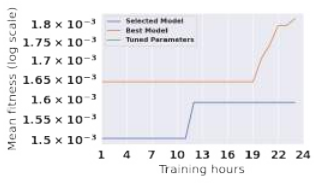
(c) Function 15



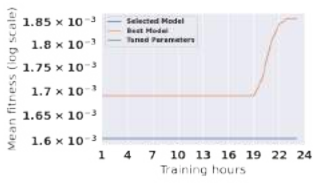
(d) Function 16



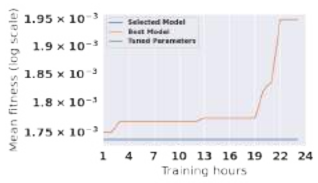
(e) Function 17



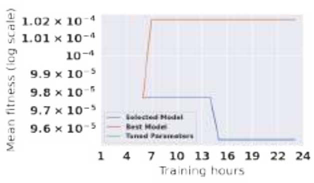
(f) Function 18



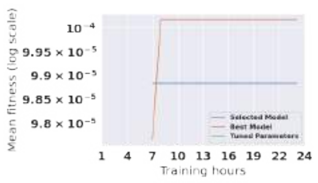
(g) Function 19



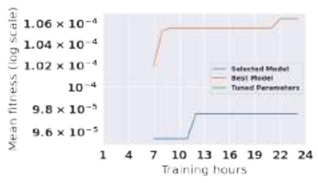
(h) Function 20



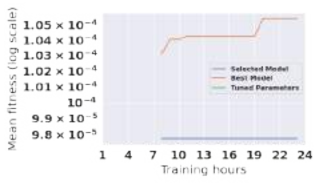
(i) Function 21



(j) Function 22

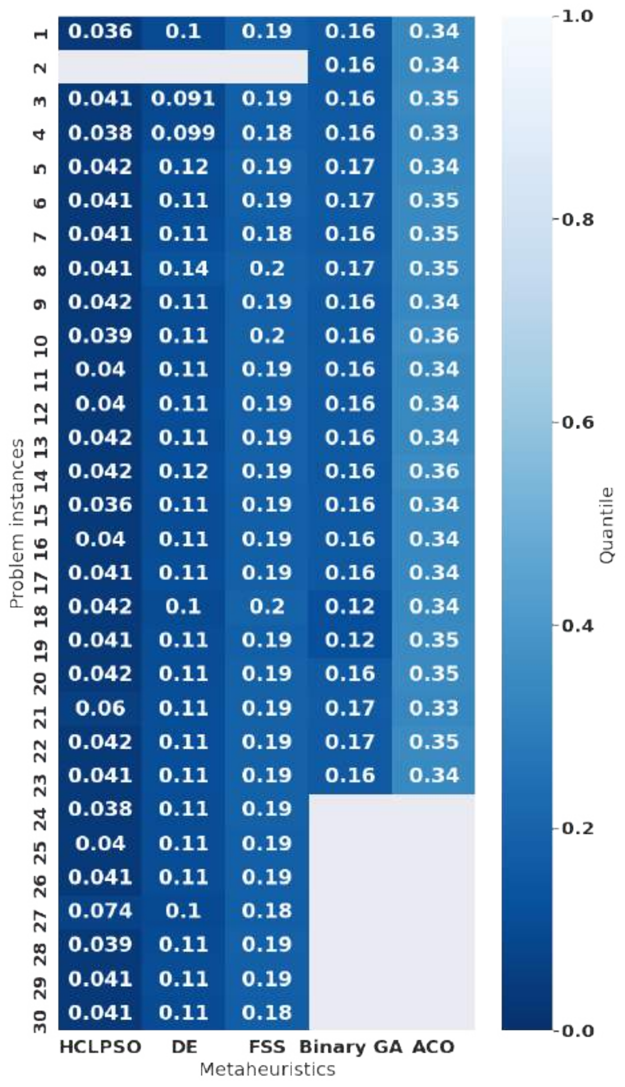


(k) Function 23



Source: Produced by the author.

Figure 79 – Quantile of the selected policies in the ranked pool of trained policies. For instance, the cell in the first row and first column shows that the selected policy to control HCLPSO in the test with function 1 is the worst performing policy among the 3.6% best policies available in the pool of trained policies for this case.



	HCLPSO	DE	FSS	Binary GA	ACO
1	0.036	0.1	0.19	0.16	0.34
2				0.16	0.34
3	0.041	0.091	0.19	0.16	0.35
4	0.038	0.099	0.18	0.16	0.33
5	0.042	0.12	0.19	0.17	0.34
6	0.041	0.11	0.19	0.17	0.35
7	0.041	0.11	0.18	0.16	0.35
8	0.041	0.14	0.2	0.17	0.35
9	0.042	0.11	0.19	0.16	0.34
10	0.039	0.11	0.2	0.16	0.36
11	0.04	0.11	0.19	0.16	0.34
12	0.04	0.11	0.19	0.16	0.34
13	0.042	0.11	0.19	0.16	0.34
14	0.042	0.12	0.19	0.16	0.36
15	0.036	0.11	0.19	0.16	0.34
16	0.04	0.11	0.19	0.16	0.34
17	0.041	0.11	0.19	0.16	0.34
18	0.042	0.1	0.2	0.12	0.34
19	0.041	0.11	0.19	0.12	0.35
20	0.042	0.11	0.19	0.16	0.35
21	0.06	0.11	0.19	0.17	0.33
22	0.042	0.11	0.19	0.17	0.35
23	0.041	0.11	0.19	0.16	0.34
24	0.038	0.11	0.19		
25	0.04	0.11	0.19		
26	0.041	0.11	0.19		
27	0.074	0.1	0.18		
28	0.039	0.11	0.19		
29	0.041	0.11	0.19		
30	0.041	0.11	0.18		

Source: Produced by the author.

APPENDIX B – PAPERS RELATED TO THIS STUDY

This section provides the papers written throughout the doctorate that are related to this study.

B.0.1 Population Size Control for Efficiency and Efficacy Optimization in Population Based Metaheuristics

Authors: Marcelo Gomes Pereira de Lacerda, Hugo Amorim Neto, Teresa Bernarda Ludermir, Herbert Kuchen, and Fernando Buarque de Lima Neto.

Abstract: This paper proposes a mechanism of dynamic adjustment of the population size of population based metaheuristics in order to balance its efficacy and efficiency. In this approach, an external trajectory based metaheuristic (MH) is used to dynamically adjust the population size of an inner population based metaheuristic. A Particle Swarm Optmization (PSO) implemented for a Compute Unified Device Architecture platform (CUDA), called CUDA-PSO, is used as inner MH, while a sequential Simulated Annealing (SA) is used as an external one. The main objective of this paper is to evaluate the SA capabilities of finding a good balance between efficiency and efficacy during the CUDA-PSO execution and to assess its adaptability to different hardwares without any prior information about the computing platform. The results show that the new approach was able to find a good balance in most cases. Also, it was observed that this approach is able to adapt its operation to different hardwares.

Keywords: metaheuristics, high performance computing, particle swarm optimization, simulated annealing, hyperheuristics.

Status: Published in 2018 IEEE Congress on Evolutionary Computation (CEC), Rio de Janeiro, Brazil, 2018, pp. 1-8, DOI: 10.1109/CEC.2018.8477792 (Qualis A1).

B.0.2 On the Learning Properties of Dueling DDQN in Parameter Control for Evolutionary and Swarm-based Algorithms

Authors: Marcelo Gomes Pereira de Lacerda, Fernando Buarque de Lima Neto, Hugo Amorim Neto, Herbert Kuchen, Teresa Bernarda Ludermir.

Abstract: This work is intended to assess the learning capability of an agent implemented with a Dueling Double Deep Q-Network in the problem of parameter control for Evolutionary and Swarm-based algorithms. The objective is to build a general parameter control method for these algorithms, that can be used for any Population Based Algorithm (PBA) to solve any numerical optimization problem, implemented for any computing platform, and is able to choose a good sequence of parameter values for the PBA, given a time budget constraint. For the experiments, an implementation of the Particle Swarm

Optimization for CUDA devices was chosen as the PBA and a set of well-known highly complex numerical minimization problems were used for the benchmark. The experiments showed that the agent is clearly able to evolve from a completely random decision policy to a fitness-minimization-oriented policy for most of the functions.

Keywords: parameter control, reinforcement learning, swarm intelligence, evolutionary algorithms, deep q-networks.

Status: 2019 IEEE Latin American Conference on Computational Intelligence (LA-CCI), Guayaquil, Ecuador, 2019, pp. 1-6, DOI: 10.1109/LA-CCI47412.2019.9036764 (Qualis B4).

B.0.3 A systematic literature review on general parameter control for evolutionary and swarm-based algorithms

Authors: Marcelo Gomes Pereira de Lacerda, Luis Filipe de Araújo Pessoa, Fernando Buarque de Lima Neto, Teresa Bernarda Ludermir, Herbert Kuchen.

Abstract: This paper presents a systematic literature review on general parameter control for evolutionary and swarm-based algorithms. General methods can be applied to any algorithm, parameter or problem, in contrast to methods that are tailored to specific applications. In this literature review, a total of 4449 studies were retrieved by the search engines and only 50 of them were selected to the extraction phase. Finally, only 15 were fully analyzed and discussed. To the best of our knowledge, this is the first literature review on such a field and one of the very few systematic reviews on parameter adjustment for those algorithms.

Keywords: Parameter control, Evolutionary algorithms, Swarm intelligence, Systematic literature review.

Status: Published in Swarm and Evolutionary Computation, v. 60, p. 100777, 2021. ISSN 2210-6502, DOI: https://doi.org/10.1016/j.swevo.2020.100777 (Qualis A1).

B.0.4 Towards a Parameterless Out-of-the-box Population Size Control for Mono-Objective Metaheuristics

Authors: Marcelo Gomes Pereira de Lacerda, Hugo de Andrade Amorim Neto, Teresa Bernarda Ludermir, Herbert Kuchen, Fernando Buarque de Lima Neto.

Abstract: We present an innovative step towards a parameterless out-of-the-box population size control for mono-objective metaheuristics. To the best of our knowledge, our approach is the first parameterless out-of-the-box parameter control for metaheuristics. It is easy to implement and to use, since it does not require extra parameters. The general idea is to increment the velocity of the population change if the best fitness stagnates, and decrement it otherwise. Then, in order to effectively change the population size, a mechanism of removal/addition of individuals inspired by the selection methods of evolutionary algorithms is executed. Our experimental results provide evidence that our controller is

not only compatible with any mono-objective algorithm and optimization problem, but that it also performs well in many scenarios.

Keywords: Population Size Control, Parameter Control, Swarm Intelligence, Evolutionary Computation.

Status: Submitted to a scientific journal.

B.0.5 A Distributed Training Process for an Out-of-the-box Parameter Controller for Metaheuristics

Authors: Marcelo Gomes Pereira de Lacerda, Fernando Buarque de Lima Neto, Teresa Bernarda Ludermir, Herbert Kuchen.

Abstract: We present an out-of-the-box methodology for training parameter-control policies for mono-objective evolutionary and swarm-based algorithms using distributed reinforcement learning algorithms with continuous action space. The proposed method is designed to be compatible with any metaheuristic to solve any mono-objective optimization problem. The main objective is to address the following shortcomings usually found in reinforcement learning-based approaches: (1) training is usually very time-consuming; (2) the approaches are restricted to discrete action spaces and (3) they require the adjustment of a large number of hyperparameters. Extensive experiments have shown that our method addresses these points very well and that it provides a new and promising direction in the field of parameter control for evolutionary and swarm-based algorithms.

Keywords: parameter control, reinforcement learning, swarm intelligence.

Status: Submitted to a scientific journal.

B.0.6 Distributed Reinforcement Learning for Out-of-the-box Parameter Control in Evolutionary and Swarm-based Algorithms

Authors: Marcelo Gomes Pereira de Lacerda, Fernando Buarque de Lima Neto, Teresa Bernarda Ludermir, Herbert Kuchen.

Abstract: The methods on parameter control for metaheuristics with Reinforcement Learning put forward so far usually present the following shortcomings: (1) parameter control algorithms are usually very time-consuming and no scalable training methods have been proposed so far; (2) RL-based controllers require the adjustment of a large number of hyperparameters and their performances can be very unstable; and (3) very limited experimental benchmarks have been used in the papers about out-of-the-box parameter control. This paper addresses these issues by proposing a methodology for training out-of-the-box parameter control policies for mono-objective Evolutionary and Swarm-based algorithms using distributed Reinforcement Learning algorithms with Population Based Training. Moreover, our method includes and exploits the benefits of distributed platforms. The experimental results in this paper show that the proposed method improves satisfactorily all the aforementioned issues.

Keywords: parameter control, metaheuristics, reinforcement learning, evolutionary algorithms, swarm intelligence.

Status: Submitted to a scientific journal.