Parameter Control Strategies of Parallel Hybrid Metaheuristics Applied to the Quadratic Assignment Problem

Jonathan Duque Gallego

Tesis de maestría presentada para al título de Magíster en Ingeniería, especialidad Informática

Director
Danny Alexandro Múnera Ramírez, Doctor (PhD) en Informática

Universidad de Antioquia
Facultad de Ingeniería
Maestría en Ingeniería
Medellín, Antioquia, Colombia
2022

Cita	(Duque Gallego, 2022)
Referencia **Estilo APA 7 (2020)**	Duque Gallego, J. (2022). *Parameter Control Strategies of Parallel Hybrid Metaheuristics Applied to the Quadratic Assignment Problem* [Tesis de maestría]. Universidad de Antioquia, Medellín, Colombia..

Maestría en Ingeniería, Cohorte XIII.
Grupo de Investigación Telecomunicaciones Aplicadas (GITA).

Biblioteca Carlos Gaviria Díaz

Repositorio Institucional: http://bibliotecadigital.udea.edu.co

Universidad de Antioquia - www.udea.edu.co

Rector: John Jairo Arboleda Céspedes.
Decano/Director Jesús Francisco Vargas Bonilla
Jefe departamento: Augusto Enrique Salazar Jiménez.

El contenido de esta obra corresponde al derecho de expresión de los autores y no compromete el pensamiento institucional de la Universidad de Antioquia ni desata su responsabilidad frente a terceros. Los autores asumen la responsabilidad por los derechos de autor y conexos.

UNIVERSIDAD DE ANTIOQUIA

Facultad de Ingeniería

Thesis

to obtain the degree of

Master's degree in engineering

Specialty: informatics

Presented by:

Jonathan Duque Gallego

Title:

Parameter Control Strategies of Parallel Hybrid Metaheuristics Applied to the Quadratic Assignment Problem

March 24, 2022

M. Danny Alexandro Múnera Ramírez *Director*

To the loving memory of my best friend Danilo
To my mother Dora
En la memoria de mi mejor amigo Danilo
A mi madre Dora

Abstract

The Quadratic Assignment Problem (QAP) is one of the most challenging combinatorial optimization problems with many real-life applications. Multiple methods have been created to solve QAP, exact and approximate methods, among others. Metaheuristics are a subset of approximative methods which have shown to be very efficient in solving QAP. Their behavior can be controlled by a set of parameters. Currently, the best solvers are based on hybrid and parallel metaheuristics. However, the design of parallel hybrid methods requires even more the fine tuning of a larger number of parameters.

The parameter setting problem (PSP) is the task of finding the correct values of the metaheuristic parameters that results in the best possible performance. It is possible to identify four main ways for solving the PSP, these are: Parameter Tuning Strategies, Parameter Control Strategies, Instance-specific Parameter Tuning Strategies and HyperHeuristics. Several methods for solving the PSP have been proposed. However, there is a need for parameter control strategies for single-solution metaheuristics, more notorious in parallel hybrid metaheuristics. To solve this problem, we have proposed PACAS, a framework to configure the **PA**rameter **C**ontrol **A**daptation for **S**ingle solution metaheuristics in a parallel hybrid solver for the efficient solution of combinatorial optimization problems. We proposed a *Java* implementation of framework J-PACAS, which implemented the functionality for solving the QAP.

Our implementation uses three popular metaheuristics applied to QAP: the Robust Tabu Search, the Extremal Optimization method and a simple Multi-start Local Search. J-PACAS also supplies three different strategies to perform the adaptation of the parameters. We present the results obtained by executing an experimental evaluation on a set of very difficult instances of QAPLIB. We explore different parameter control strategies, with different parallel configurations (independent or cooperative). We compare the best J-PACAS configuration identified in the experimental evaluation against a competitive state-of-the-art parameter control method, finding that our implementation presents a similar performance in small instances and a better performance in hard instances of the QAPLIB benchmark.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The Quadratic Assignment Problem (QAP) is defined as the assignment of facilities, or tools, to exactly one location and vice versa. The distance between the locations and the flows between facilities is known, the problem is to find the minimum cost associated to assign a facility in one location; the cost is the sum of all the products between flows and the distance [Kaufman and Broeckx, 1978]. QAP is one of the best known combinatorial optimization problems and one of the most difficult (since it is NP-hard), it was presented by Koopmans and Beckmann in 1957 as a mathematical model for the location of indivisible economic activities [Koopmans and Beckmann, 1957].

Real-life applications can be modeled as QAP [Wu et al., 2012], such as electronic chipset layout and wiring, scheduling, process communications, turbine runner balancing, data center network topology, among many others [Commander, 2005, Bhati and Rasool, 2014]. In general, one can find applications of QAP in multiple areas, such as archeology, statistics, economics, chemistry, logistics and electricity, among others. Even problems like the traveling salesman (TSP) can be formulated as a QAP problem [Loiola et al., 2007a]. Other types of QAP transformable problems can be found in [Burkard, 1984, Burkard et al., 2012].

1.1 Solution Methods for QAP

QAP can be solved using exact and approximative methods. Exact methods consider the entire search space: either explicitly by exhaustive search or implicitly, by pruning some portions of the search space that have been detected as irrelevant for the search. However, the QAP presents a "combinatorial explosion" i.e., the size of the search space grows exponentially with relation to the size of the instance (number of variables, location or facilities). For this reason exact methods can not solve QAP instances with size larger than 35 in a reasonable time [Loiola et al., 2007b]. In contrast, approximative methods efficiently explore only some portions of the search space, obtaining a good sub-optimum solution (local optimum) in a reasonable time.

Metaheuristics are a subset of approximative methods which have shown to be very efficient in solving QAP. Metaheuristics are algorithms usually inspired by nature or physical principles. These methods usually make decisions to efficiently explore the search space of the problem and use randomness in their behavior [Boussaïd et al., 2013]. Some examples of metaheuristics are genetic algorithms, local search, or simulating annealing.

Metaheuristics consider two main working principles, intensification and diversification. Intensification refers to the method's ability to deeply explore a promising region of the search space, while diversification refers to explore different regions of the search space. By design, some metaheuristics methods are better at intensifying the search while others are so at diversifying it. Nonetheless, a set of parameters controls the behavior of most metaheuristics. A fine tuning of these parameters is therefore crucial to achieve an effective trade-off between intensification and diversification, and hence good performance in solving a given problem. Unfortunately, selecting the best-performing set of parameters is usually a hard task. This process is even more difficult because the best parameters values are different for different problems and even for different instances of the same problem, as demonstrated by the Non-Free-Lunch theorem [Wolpert and Macready, 1997].

Each metaheuristic has its own strengths and weaknesses, which may vary according to the problem or even to the instance being solved. The trend is thus to design hybrid metaheuristics, which combine diverse methods in order to benefit from the individual advantages of each one [Blum et al., 2011]. This increases the number of parameters (parameters of individual metaheuristics and new parameters to control the hybridization). The design and implementation of a hybrid metaheuristic is a complex process; setting the resulting parameters to reach the best performance is also very challenging.

Despite the good results obtained with the use of hybrid metaheuristics, it is still necessary to reduce the processing times needed for the hardest instances [Saifullah Hussin, 2016]. One of the most plausible options entails parallelism [Crainic and Toulouse, 2010]. In parallel metaheuristics, one can have multiple instances of the same (or different) metaheuristics running in parallel, either independently or cooperatively, through concurrent process communications [Caniou et al., 2014]. Parallelism not only helps to decrease processing time but also is a mean to easily implement hybridization.

The creation of parallel hybrid methods requires the fine tuning of a larger number of parameters, since more metaheuristics (of different types) are involved. Moreover, the configuration of the parallel interaction itself (communication between the methods) involves yet another set of parameters which need to be adjusted. Tuning this increasing number of parameters makes it even more difficult to find the appropriate setting for the algorithm to behave optimally. Automating the task of finding good parameters is thus desirable and has attracted significant attention from researchers.

1.2 Setting Metaheuristic Parameters

It is possible to identify three specific strategies focused on solving the task problem of setting the metaheuristic parameters: *parameter tuning*, *parameter control* [Huang et al., 2020] and *instance-specific parameter tuning* [Calvet et al., 2016]. In parameter tuning (off-line tuning) the set of parameters are defined before applying the algorithm to a specific problem instance(s) (static definition of parameters). Several strategies for parameter tuning of metaheuristics have been proposed [Birattari and Kacprzyk, 2009, Hutter et al., 2009, 2011]. In contrast, parameter control strategies (online-tuning) adapt the values of the controlled parameters during the algorithm execution (dynamic/reactive adaptation of parameters). The idea is to find the best parameters setting during the solving process, using some mechanism to alter the parameter values according to the observed algorithm performance. The instance-specific strategy considers a combination of the two previous methods (parameter tuning and parameter control) with the aim of having for each specific instance of a problem, one set of static parameters values, few papers can be found with this approach [Ries and Beullens, 2015, Calvet et al., 2016, Dobslaw, 2010a].

Parameter tuning can be seen as a pre-process pass which is executed before the solving procedure in order to determine the adequate values for parameters. This does not affect the implementation of the solver. Unlike parameter control strategies, which has to be implemented in the kernel of the solver, as part of it. Parameter tuning strategies may appear easier, but when the number of parameters becomes large, it is hard to use in practice, e.g., within a parallel hybrid method. Indeed, it usually requires many runs to identify the best parameter settings, making this a time-consuming process. These methods are often limited by the number of parameters and the computational power available. In that case, parameter control strategies emerge as a viable solution to deal with the high complexity of current solvers (hybrid and/or parallel).

Regarding the instance-specific parameter tuning strategies, these avoid the need to modify the metaheuristic method and try to minimize all the computational effort that parameter control strategies require adapting parameters. Additionally, these define parameters values for each specific instance since looking for one fixed set of parameter values for an entire set of instances with different characteristics is likely not performing well as parameter tuning does [Ries et al., 2012].

There is another way to setting the parameters (a fourth way), with a more general purpose, Hyper-heuristics. These methods are part of a novel research approach in which a high level strategy selects or generates the best metaheuristics with their respective parameters and solutions acceptance criteria. This approach is used with the aim of having more general methods, not designed for a single problem or for a few instances of a problem [Burke et al., 2019].

As shown in the analysis of the state-of-the-art, despite the strategies available to solve the problem of setting the parameters of a metaheuristic, there is a need for parameter control strategies for single-solution metaheuristics, more evident in

parallel hybrid methods. The main contribution of this thesis is *to propose a general framework to configure the parameter control strategy for single-solution metaheuristics in a parallel hybrid solver, for solving combinatorial optimization problems.*

1.3 Thesis Goals and Contributions

In this research, we study the methods that have been proposed for the resolution of the parameter setting problem. By performing this study, we have noticed the gap that exists for parameter control strategies (PCS) methods in single-solution metaheuristics. This gap is even more notable for parallel hybrid metaheuristic methods.

The contributions of our research are the following:

- We propose **PACAS**: a general framework to configure the **PA**rameter **C**ontrol **A**daptation for **S**ingle solution metaheuristics in a parallel hybrid solver, for solving combinatorial optimization problems (COP). PACAS aimed at increasing the performance of parallel method based on several parameter control strategies through the analysis of the behavior of single-solution metaheuristics. The framework allows to customize the parameter adaptation strategy in order to have a trade-off between intensification and diversification in the search process. The hybrid behavior is granted by the use of different types of metaheuristics and by cooperation mechanism through an intra-team communication.

- In order to validate our approach, we present J-PACAS: an implementation of PACAS in the *Java* programming language. Our implementation provides three different metaheuristics and three kinds of teams to perform different parameter adaptation strategy. We provide the code to support one of the most difficult COPs, the Quadratic Assignment Problem (QAP). J-PACAS implementation is available in open source code through a git repository. It is implemented in Java 11 using the `ForkJoinPool` and `AtomicType` classes to handle the parallelism in a shared memory model. J-PACAS can be executed on a multicore parallel platform.

- We tackle one of the most difficult problems in combinatorial optimization, the Quadratic Assignment Problem. We consider the set of very hard QAP problems, the QAPLIB benchmark. We make a contribution exploring different parameter adaptation strategies, with different parallel configurations (independent or cooperative). Finally, we make a comparison with the most similar work in the state-of-the-art. Partial results of this experimentation were published in the International Conference on Optimization and Learning [Duque et al., 2021]

Finally, this document is organized as follows:

Chapter 2 explains the concepts, definitions and main related principles of metaheuristics and their parameter settings to understand this thesis.

Chapter 3 is dedicated to analyze the state of the art related to the strategies to solve the parameter setting problem for metaheuristic methods. This chapter contains an analysis of the most relevant characteristics of the strategies, advantages and disadvantages. According to this analysis, we further present our taxonomy that covers all the strategies reviewed.

Chapter 4 introduces the PACAS framework that we propose, detailing the design aspects for each part.

Chapter 5 presents an implementation of the PACAS framework made on the *Java* programming language, J-PACAS. We provide guidelines to extend the framework functionality and to configure all its parameters.

Chapter 6 describes the experimental evaluation to validate the functionalities of our proposed framework. We compare the performance of seven J-PACAS configurations on the most difficult instances of QAPLIB. We also make an analysis of the winner metaheuristic and its configuration. Finally, a comparison with the most similar work in the state-of-the-art is done.

Chapter 7 concludes this thesis and presents some perspectives about future research directions.

Chapter 2

Background

This chapter includes a formal definition of the Quadratic Assignment Problem (QAP), analysing its complexity. Then, it presents a brief description of metaheuristic methods used to solved QAP, as well as a presentation of the research trends related to this topic. Finally, we present a formulation of the parameter setting problem and the approaches to solve this problem.

2.1 Quadratic Assignment Problem

The Quadratic Assignment Problem (QAP) was first proposed by Koopmans and Beckmann in 1957 [Koopmans and Beckmann, 1957] as a model for a facilities location problem. This problem consists in assigning a set of n facilities to a set of n locations, while minimizing the cost associated with the *flows* of items among facilities and the *distance* between them. Let F be the flow matrix, where f_{ij} is the flow between facilities i and j, let D be the distance matrix, where d_{kl}, is the distance between the locations k and l. The goal is to find an optimum assignment of facilities to locations at minimum total cost, which is defined as the sum of all products between flows and distances [Kaufman and Broeckx, 1978]. Equation 2.1 presents a formulation of QAP, where $\phi(i)$ is the location which facility i is assigned to and $d_{\phi(i)\phi(j)}$ is the distance between locations $\phi(i)$ and $\phi(j)$.

$$\min \sum_{i=1}^{n} \sum_{j=1}^{n} f_{ij} * d_{\phi(i)\phi(j)} \tag{2.1}$$

QAP has a computational complexity that classifies it as an NP-hard problem [Sahni and Gonzalez, 1976]. Technology has advanced in a way that the processors speed has increased, however QAP remains computationally difficult to accurately solve it when the number of facilities and locations is greater than 35. It is at this point that methods with metaheuristic approaches produce high quality solutions in reasonable times [Dokeroglu and Cosar, 2016].

2.2 Metaheuristics

Metaheuristic methods are a type of approximate methods used to solve difficult optimization problems, most of them are inspired by nature (based on some principles of physics, biology or ethology); theses methods use stochastic components (use randomness in their behavior) and have several parameters that must be adjusted depending on the problem in question [Boussaïd et al., 2013].

Metaheuristics operate on two main working principles: intensification and diversification, also call exploitation and exploration respectively. On the one hand the former refers to the method's ability to explore more deeply a promising region of the search space and does not explore the whole set of solutions, providing at the end of the search the best solution of that region, called optimum, which is not necessary the global optimum of the whole search space; the optimum of this region is called local optimum. On the other hand, diversification refers to the exploration of different regions of the search space. Diversification helps to escape from local optima, guiding the search to other promising regions of the solution space and avoiding stagnation. The hybridization of metaheuristics helps to have a balance between intensification and diversification, by combining the characteristics of different methods.

We can classify metaheuristics as single-solution metaheuristics and population metaheuristics. The main difference between these two categories depends on the number of solutions used during the execution of the algorithm. A single-solution metaheuristic starts with an initial solution and at each step of the search the current solution is replaced by another (often the best) solution found in its neighborhood [Alba et al., 2013]. The neighborhood is a set of solutions obtained by making small disturbances or changes to one solution [Yagiura and Ibaraki, 2002]. The single-solution metaheuristics allow to quickly find an optimum solution locally, so these are good at intensifying the search space. Population-based metaheuristics make use of a population of solutions. In this case, the initial population is generated randomly (or created with a greedy algorithm), and then improved through an iterative process. At each generation of the process, all (or part) of the population is replaced by newly generated individuals (often the best ones). These methods are commonly used for exploration, since they work with many solutions, producing a larger search at the solution space, and trying to make the search more diverse [Alba et al., 2013].

Some of the best known metaheuristics are described below:

- Local Search (LS): LS is one of the oldest and most frequently used metaheuristics. LS starts from an initial solution and repeatedly replaces it with better solutions within its neighborhood, this method finishes when there are no better solutions in the neighborhood of the current solution, ending the search in a local optimum [Yagiura and Ibaraki, 2002].

- Simulated Annealing (SA): In 1983, Kirkpatrick et al. introduced the concept of Simulated Annealing. This algorithm is based on the method of cooling a

material, a technique called annealing. In this process, if the material is cooled down too fast, the atoms have no chance to produce a powerful structure and settle randomly, resulting in a brittle metal; but if the temperature decreases slowly, the atoms have more time to build strong crystals increasing the quality of the product [Dokeroglu and Cosar, 2016]. The SA algorithm begins with an initial solution, then this solution is transformed into another by introducing small disturbances or changes. The algorithm replaces the current solution when the resulting solution is better than the current one, ending the search when there are no improvements (local optimum). SA uses a probability function to accept worse solutions, this function simulates the way the temperature decreases in the cooling of the material. This function will allow, with a high probability, worse solutions and, as the execution of the algorithm advances, this probability decreases allowing only improving solutions [Dowsland and Diaz, 2003].

- Tabu Search (TS): The Tabu Search method refers to the use of adaptive memories and special problem-solving strategies, called intelligent strategies, which differentiate TS from a LS. The idea is to memorize within a structure the elements that for the LS will be forbidden to use and thus avoid staying in local optima. TS searches within the neighborhood for the best solution but does not visit the solutions of previous neighbors if they have been visited before or have been marked as prohibited solutions or "tabu" solutions [Dokeroglu and Cosar, 2016].

- Genetic Algorithm (GA): GA is a population algorithm, it has a direct analogy with nature, in this case biological evolution. GA works on a population of individuals, where each one represents a solution. Each individual is assigned a fitness, according to how good is the solution for the problem, i.e, the fitness is the value for the objective function [Beasley et al., 1993]. This population evolves by applying crossing and mutation operators to some individuals. The best set of new individuals replaces the initial population (elitist policy). The algorithm finishes when a given number of iterations have been performed or when the population converges.

- Ant Colony Optimization (ACO): ACO is inspired by the behavior of ants to find the shortest path from their nest to a food source. Ants mark their way to the food source by placing pheromones, at the end, the path with the most pheromones persists. The idea of the ACO as an algorithm is to start from an initial population of solutions (representing ants) and, through an iterative process, the ants that find good solutions guide the following ants by using a memory, this memory will be the pheromone information saved by the ants when they find good solutions [Merkle and Middendorf, 2001].

In addition to the metaheuristics that are presented, there are others like, Pilot Method [Voss et al., 2005], Variable Neighbourhood Search [Hansen et al., 2003],

Table 2.1: Metaheuristic parameter and its influence (note that MH stands for Metaheuristic type).

MH	Parameter name	Parameter type	Influence
TS	Tabu duration factor	integer	Small value promotes intensification. Large value promotes diversification.
	Aspiration factor	integer	It is used as a solution cost exception to take good solutions.
SA	Temperature function	function	It controls the acceptance criteria for new solutions. It avoids the stagnation of the search, making it more diverse.
GA	Population size	integer	A small value promotes intensification. A large value promotes diversification.
	Crossover operator	operator	It helps to diversify the search, guiding the exploration toward different neighborhoods.
	Mutation probability	decimal	Small value promotes intensification. Large value promotes diversification.

Greedy Randomized Adaptive Search Procedures (GRASP) [Resende and Gonzáles Velarde, 2003], Scatter Search [Martí and Laguna, 2003], Particle Swarm Optimization (PSO) [Merkle and Middendorf, 2001], among others.

Each metaheuristic type has some intrinsic parameters that guide the search to behave in one way or another. While there are methods that are better for intensification or diversification, adjusting those parameters offer a trade-off between exploitation or exploration search. Table 2.1 shows some metaheuristic methods with their parameters and its influence over the search. In addition to these intrinsic parameters, there are parameters common to most methods. For example, the stopping criterion and the type of neighborhood to explore. It is important to note that even for some parameters it is necessary to define other "internal" parameters.

There is no theory to determine the most adequate metaheuristics for a problem or the optima values for its parameters, so usually different strategies have been proposed, such as consulting an expert opinion, extracting values from state-of-the-art works, developing a design of experiments or even invoking to the developer's intuition. This is an import topic to take into account, given that the quality of the solutions will depend on the choice of the metaheuristics and on the values of its parameters [Turky et al., 2018].

The *configuration* of the metaheuristic parameters is a fundamental part of the algorithm design. This *configuration* can also include changing the order in which the *components* of a metaheuristic are executed, where a *component* is a specific and unique part that defines the algorithm and each one may have one or more parameters that determine its functionality. Examples of components are a local search operator in a Variable Neighbourhood Search, a tabu list in a TS, a crossover operator or a mutation operator in an GA. The configuration of these components is called the *control flow* [Sevaux et al., 2015].

Determining the *control flow* of the algorithm is a task for the algorithm designer. There is no a guide for the best order or to choose the best components for a method [Sevaux et al., 2015]. In general, methods usually have a default order and available components. The practitioners work with this default mode, since configure it, is long and fastidious task, usually done with trials and errors methods. One question arose, is there an optimum configuration for achieving the best performance?. While this question is of interest to the target community, it is broader than our research question and is beyond our scope.

2.3 Hybrid Metaheuristics

Having a equilibrium between intensification and diversification is a hard task [Beasley et al., 1993], with this aim hybridization have been used. The propose is adding up metaheuristic strengths and balancing their weaknesses. In recent years, the research on hybrid metaheuristics has notably increased [Bhati and Rasool, 2014], positioning hybrid methods as one of the best performing solvers for tackling many hard problems. Particularly for QAP, hybrid methods provide outstanding performances for difficult QAP instances.

The basic idea about hybridization of metaheuristics is the design and implementation of algorithms that, through the combination of different metaheuristics retains the best features that each one has to offer. Although the way as metaheuristics are hybridized varies according to the type of problem or the type of method, several research works have been proposed to define criteria that allows the development of a hybrid metaheuristic, for example [Blum et al., 2008].

2.4 Parallel Metaheuristics

The parallelization of metaheuristics is another strategy that has been used simultaneously with hybridization. At the early days, the parallelization has been done with the traditional idea of accelerating the execution time of the operations of the algorithm, focusing on operations that are more expensive in processing time. Later, it has been used to distribute tasks or sets of tasks on the available processors, tasks that result from the decomposition of the total computational load of the algorithm.

Four major strategies can be identified in the parallelization of metaheuristics. The first strategy is the decomposition of the low level computer tasks without modifying the original algorithm but accelerating its execution, for example by parallelizing the execution of the neighborhood assessment in a single-solution metaheuristics or the population assessment in a population-based metaheuristics. The second strategy is the decomposition of the search space into smaller portions and the assignment of one or more processors for exploration using one or more metaheuristics. To make the search space decomposition, special techniques are used for its partitioning and

special techniques to assemble solutions from the solutions found in the exploration of each partition. The third strategy is the independent multiple search which consists of executing several self-contained searches, each with a different initial solution and with no exchange of information between them. The searches have a starting point that is usually random and to the final solution no partitioning and composition technique is applied. Finally, the cooperative multiple search is similar to the independent one with the difference that the metaheuristics that are executed simultaneously resort to the exchange of information during their execution [Codognet et al., 2018]. Cooperative parallelization not only improves the processing times but also open a bunch of possibilities to create new hybrid algorithms.

Parallel hybrid metaheuristics often have many parameters which modify the algorithm behaviour. Setting these parameters has a heavy influence on the performance of the method, however, finding the optima values for these parameters is usually a hard task [Hutter et al., 2009]. Using hybridization and parallelism makes this task even more difficult for mainly two reasons: First, hybrid metaheuristics inherit the parameters of each "low level" metaheuristic, so one needs to find the setting of more parameters, since a parameter configuration for one algorithm usually is not suitable for another. Second, cooperative parallel strategies also require parameters to define their behaviour, e.g., for determining how frequently metaheuristics should interact or how each metaheuristic has to use the received information.

2.5 Setting Metaheuristic Parameters

The metaheuristic parameters can be of different kinds, as show in table 2.1. There are integers, decimals, functions, operators, and others. These can be classify in two groups, numerical and categorical (or non-numerical) parameters [Huang et al., 2020]:

- Numerical: Corresponds to all those parameters that are real or integer numbers, and they are typically in a range of values.
- Categorical: Refers to the remaining parameters that control the behavior of a metaheuristic, these are functions, mathematical expressions, operators, or mechanisms. One example is the crossover operator in the GA algorithm.

The Parameter Setting Problem (PSP) is the task of finding the parameter values for a metaheuristic that results in the best possible performance across the given problem instance(s). This problem can be briefly described as: given an algorithm method (to be parametrized), one instance or a set of problem instances, and a free set of parameters to choose, the purpose of parameter configuration is to resolve the problem of finding such values that optimizes the objective function of the algorithm over the given problem instance(s).

Let M the metaheuristic algorithm to parameterize, P the parameter space values, a set of problem instances I and a performance metrics m_p that measures the performance of M across I for a given configuration p ($p \in P$). The problem is therefore to find a configuration $p* \in P$ that optimizes the performance of M on I according to the metrics established m_p [Huang et al., 2020]. The PSP is hence itself a combinatorial optimization problem (COP) and, in general, NP-hard [Kern, 2006] and it is often called meta-optimization [Pedersen, 2010].

Three dedicated strategies to solve the PSP can be found, these form a range of strategies categorized into Parameter Tuning Strategies (PTS), Parameter Control Strategies (PCS) [Huang et al., 2020] and Instance-specific Parameter Tuning Strategies (IPTS) [Calvet et al., 2016]. There are others strategies not dedicated to solving the PSP, however, as part of its purpose solves it, these are HyperHeuristics methods (HH) [Burke et al., 2019]. In the next subsections we define each category.

2.5.1 Parameter Tuning Strategies (PTS)

Eiben et al. define Parameter Tuning Strategies as *"The commonly practiced approach that amounts to finding good values for the parameters before the run of the algorithm and then running the algorithm using these values, which remain fixed during the run"* [Eiben et al., 1999]. The aim is to find a *priori* the best set of values for the parameters. With a preliminary analysis, standard values for the parameters can be recommended for future executions. However, such recommendations should not be generalised to all classes of problems.

2.5.2 Parameter Control Strategies (PCS)

Parameter Control Strategies (also known as online setting) is an approach that eliminates the parameter values analysis step carry out in PTS, therefore the setting occurs during the optimization process (algorithm running) [Parpinelli et al., 2019]. In this methodology is required adequate control strategies for the controlled parameters, these strategies change or adapt relevant parameter values during the run [Huang et al., 2020]. The control strategies usually make use of information gathered during the execution of the algorithm as a feedback, based on this feedback, the parameter are adapted dynamically [Calvet et al., 2016].

2.5.3 Instance-specific Parameter Tuning Strategies (IPTS)

Instance-specific Parameter Tuning Strategies (IPTS) the parameters remain fixed during the run, similar to PTS. Therefore, there is a design phase that precedes the algorithm execution to find the parameter values. In the design phase, a representative set of instances is first investigated. This instance set is used to design an efficient tuning method (or model) that can return instance-specific parameter values for each specific problem instance, rather than aiming to obtain one set of parameter values for

all instances. The tuning method is done based on measurable instance characteristics. Further, in the execution phase, an instance is examined by the tuning method in order to return a set of instance-specific parameter values [Ries et al., 2012].

The IPTS approach is a new research area, hence the published works are few and sometimes the differences with PTS are diffused. The differences are not yet clear because in both strategies the parameters remain fixed. One can find in the literature papers categorized as PTS, but those match perfectly under the definition of IPTS, for example [Pavón et al., 2009].

2.5.4 HyperHeuristics (HH)

HyperHeuristics (HH) present another way to face the problem of metaheuristic parameter setting and they have some similarity to PCS. This is a novel research approach, it appears in 2001, with the purpose of selecting or generating the best metaheuristics with their respective *parameter settings* and acceptance criteria to face a problem. HH are methods to manage the *control flow* of an algorithm by applying some strategies to select or generate the best component order, with the aim of having the best parameter settings and configuration for a the method. This approach is used with the idea of having more general methods, not only designed for a single type of problem or for a few instances of a problem. HH usually makes use of learning strategies outside or inside the execution of the algorithm [Burke et al., 2019].

A hyperheuristic is also known as a heuristic to select/generate metaheuristics, heuristic known as a high-level heuristic. The set of available metaheuristics or metaheuristic components are known as a low-level heuristics. A high-level heuristic works on the set of metaheuristics or metaheuristic components rather than on the search space of solutions of the given problem, it is independent of the type of problem. A HH aims to automate the design and adaptation of metaheuristic methods to address computational search problems, in our case COPs. The motivation is raising the level of generality in which search methodologies can work [Burke et al., 2019].

These methods are learning algorithms, which use feedback during the search process. According to the way of feedback during this learning, one can distinguish between online and offline learning. In online learning hyperheuristics, learning takes place while the algorithm is solving an instance of a problem (similar to PCS). While in offline learning hyperheuristics, the idea is to gather knowledge in the form of rules or programs, from a set of training problems, which will help to solve instances of unknown problems [Burke et al., 2013].

Chapter 3

State of the Art

The problem of setting the best set of parameters is called the Parameter Setting Problem (PSP). The PSP is gaining a lot of interest among researchers and practitioners, since setting parameters by hand or by experience is not the best option. Classical methods like using design of experiments to identify a good set of parameters is computational expensive and time consuming for the designer. This is one reason why parameter setting studies have not been extensively developed [Dobslaw, 2010b], but since the mid-1990s it is an issue that is taking great importance [Calvet et al., 2016] and the scientific community started to treat it formally, as shown by Ferro's work [Ferro et al., 1996b].

From the literature, it is possible to identify four main ways for solving the PSP, these are: Parameter Tuning Strategies (PTS), Parameter Control Strategies (PCS), Instance-specific Parameter Tuning Strategies (IPTS) and HyperHeuristic (HH). These methodologies form a group of strategies, which perform different functionality, from the "limited" ability to change only the parameters, to the "complete" ability to change the whole algorithm.

The literature about parameter settings for metaheuristics is diverse, there is no review that covers completely the four ways to solve the PSP. The most studied strategy is PTS, and several taxonomies can be found for this approach [Huang et al., 2020, Dobslaw, 2010b]. For PCS, one can also find classifications made only for population algorithms, most of them for evolutionary algorithms [Parpinelli et al., 2019, Zhang et al., 2012, Eiben et al., 2007]. In addition, some publications consider the PSP in a summarized but not detailed or exhaustive way [Sevaux et al., 2015, Stützle and López-Ibáñez, 2019]. For the HH strategy the situation is different. Burke et al. [2019] present a recent classification and taxonomy. It is important to remember that these kinds of works consider the total manipulation of the method (either generation or selection) not only the setting of the parameters.

The intention of our review is to bring out the most relevant methodologies that exist for setting the parameters of metaheuristics when solving hard optimization problems and show how researchers have classified them. We present a detailed description of each strategy, pointing out main characteristics, advantages and disadvantages and

some of the most relevant papers that have been published. Finally, a complete taxonomy is presented and discussed. In order to achieve the proposed taxonomy, we consider the aforementioned reviews and many others who make interesting contributions to this field.

3.1 Parameter Tuning Strategies (PTS)

As mentioned above, PTS can be defined as the procedure to find good values for the parameters before the execution of the metaheuristic algorithm. PTS methods allow the user to find a good set of parameters for running the algorithm with these values, which will remain fixed throughout the execution. This strategy can be seen as a preprocess phase or as an off-line parameter setting. Dobslaw called this step *algorithm configuration* [Dobslaw, 2010b], because is the configuration before trying to solve the problem. Meanwhile, A. Fialho, in his Ph.D. thesis, called these strategies as *External Parameter Tuning* since it is a process done outside the algorithm core [Fialho, 2010].

From the beginnings of metaheuristics, parameter setting was done by hand. When more metaheuristics methods were created showing better results, one common strategy was to use parameters' values reported in the literature as a starting point for finding a good set of parameters in new algorithms. This is indeed one of the most common strategies that has been used by most researchers [Sevaux et al., 2015].

However, there is a problem. On the one hand, the idea of taking parameters from similar works is not suitable in all cases. This is a blind process which not considers the particularities of new algorithms or instances. On the other hand, the task of testing many parameters values via trials and errors methods for finding the best ones can be very time-consuming and tedious. That is why more elaborate tools were created. These tools aimed at finding the best initial parameters of a solver, in order to yield (near-)optimum algorithm performance. One of the pioneering strategies was the use of the classic Design of Experiments (DoE), then followed by a large number of PTS methods, such as EGO, F-Race, REVAC, ParamILS, SPO, SMAC, SKO, among others. We describe below the most known methods.

As we found in the analysis of the state-of-the-art for PTS methods, parameter tuning methods are a large part of the works that has been proposed to solve the PSP. Some methods have its origin in the Design of Experiments (DoE). DoE methods have been refined and extended for proposing new ones since the 90s, e.g, EGO, SPO, SKO, among others. Metaheuristic methods are also used to find the right parameters, an example is GGA, Calibra and HORA. Also, racing techniques are a common strategy, which aim to discard the irrelevant parameters during the procedure and keep the ones who are the better, for instance, F-Race, Iterative F-Race, HORA and SMAC.

In the following, we present a definition of each identified technique for solving the PSP in PTS methods:

- Brute-Force: The most intuitive and easiest method for solving the tuning problem is the brute-force. This method consists in allocating an equal share of

computational power to each candidate configuration and to perform the same number of experiments on each part. The configuration with the best estimated performance is considered the optimum parameter setting. To reach good parameter values, high amount of executions are necessary and a large number of parameters must be tested, being this the main weakness of the brute-force method.

- DoE: Design of Experiments is a technique for conducting experiments with the objective of identifying the causes of the behavior of a problem and get conclusions. These conclusions are drawn on the basis of statistical methods [Fisher, 1936]. Usually, the full factorial design is the most applied for PTS. In this approach, the researcher takes a small number of the problems from the entire problem set (representative instances). Then an experiment is carried out with several values of the parameters, parameters act as factors with various levels, taking into account lower and upper bounds and values in the middle. This process allows the user to select good parameter settings for each problem. An example of DoE using a full factorial design for setting parameters is presented in [Coy et al., 2001]. Other kinds of DoE techniques had been applied, like block design, and response surface optimization.

- Revac: Relevance Estimation and Value Calibration of Parameters (REVAC) method was proposed in 2006. This method is used to calibrate the values of parameters of any evolutionary algorithm. The method works on distributions over parameter values and uses the Shannon entropy to estimate the relevance of a parameter [Nannen and Eiben, 2006]. REVAC is an iterative algorithm that consists of estimating the distributions of promising parameter values for each parameter within the configuration space, and then generating parameter configurations by drawing values from these distributions [Huang et al., 2020]. One important weakness of REVAC is that this method cannot handle categorical parameters.

- ParamILS: This method is one of the best-known methods for algorithm configuration. ParamILS is an automatic tuning method proposed by Hutter et al. [Hutter et al., 2009] in 2009. The authors present two versions: BasicILS and FocussedILS. Both versions use iterated local search (ILS) techniques in order to guide the search towards promising areas in the search space of the parameters. BasicILS is the simplest approach that evaluates every parameter configuration by running it on the same training problem instances using the same random number of seeds. FocussedILS is a variant of BasicILS that adaptively makes a variation in the number of training samples considered from one parameter configuration to another. ParamILS can use the so-called adaptive capping mechanism and is able to tune both numerical and categorical parameter. The software

of ParamILS is implemented in the Ruby programming language [1].

- EGO: Efficient Global Optimization (EGO) is an algorithm for the creation of a response surface model for black box functions. It combines the predictive model (Kriging or Gaussian process model), i.e., design and analysis of computer experiments (DACE) [Sacks et al., 1989], for diversification and a branch and bound-based phase for intensification. The EGO algorithm fits a DACE model with a set of initial points specified for the experimental design. The hardware requirements for this method are low. EGO is restricted to non-stochastic algorithms [Jones et al., 1998]. The SPO and SKO methods extended EGO.

- SPO: The Sequential Parameter Optimization (SPO) method is an extension of EGO. This method was presented by Thomas BartzBeielstein, Christian Lasarczyk, and Mike Preuss in 2005 [Bartz-Beielstein et al., 2005]. The main purpose is to determine improved parameter settings for optimization algorithms to analyze and understand their performance. This makes use of Kriging models to perform the optimization, creating a response surface model. With this model, new set of design points (parameter settings) are generated and tested. The most promising points, which have the highest expected improvement, are chosen as new candidate configurations through an iterative process. The corresponding software package will be referred to as SPOT, an acronym which stands for *sequential parameter optimization toolbox*. SPOT package for R can be downloaded and used for free. Unfortunately, SPO only supports numerical parameters and only optimizes an algorithm for a single instance [2].

- SKO: Another extension of EGO, Sequential Kriging Optimization (SKO) was proposed by Huang et al. [Huang et al., 2006]. The method is based on a kriging metamodel that provides a global prediction of the objective values and a measure of prediction uncertainty at every point. SKO is formulated to extend the EGO scheme to stochastic systems. SKO restricts the user to optimize continuous parameters. It is a sequential approach; the model is becoming updated at each iteration until a certain quality or time-bound is reached [Dobslaw, 2010b].

- F-Race: It is a racing algorithm that uses the Friedman test (two-way analysis of variance by ranks), a non-parametric statistical test [Birattari et al., 2002]. In this method, all the configurations take part in a race, some are discarded based on the statistical study and those who pass the test continue the race. In each evaluation round of the candidate configurations, the non-parametric Friedman test is used as a family-wise test to check whether there is evidence that at least one of the candidate configurations is significantly different from others in terms of performance measures. The original version has the drawback

[1] ParamILS software available at `http://www.cs.ubc.ca/labs/beta/Projects/ParamILS/index.html`

[2] SPOT package available at `https://cran.r-project.org/web/packages/SPOT/index.html`

that only works for a handfull of parameters. The reason is that experimental design is expensive. There is a free available extension in R language called *race*.

- Iterated F-Race: To alleviate the problem of F-Race for the tuning problem when there are a large number of parameters and/or each parameter has a wide range of possible values, Prasanna Balaprakash, Mauro Birattari and Thomas Stützle [Balaprakash et al., 2007] proposed the iterative application of F-Race. Iterated F-Race is an iterative procedure in which each iteration consists in the definition of a probability measure over the parameter space that is available at the moment, using promising configurations obtained from the previous iterations. Then, the method generates configurations according to the newly defined probability measure and finally applies a standard F-Race on the generated configurations. For the probability measure, it is used a d-variate normal distribution parameterized by mean vector (using surviving configuration from the previous iteration) and covariance matrix. This measure plays a crucial role in biasing the search towards regions containing high-quality configurations. Iterated F-Race has become one of the most competitive PTS [Huang et al., 2020]. It can deal with both numerical and categorical parameters. An R package is available called *irace* [3].

- Calibra: This method employs statistical analysis techniques (for exploration) and a local search (for exploitation). Calibra is a procedure to create a systematic way to find good parameter settings within a specified range of values [Adenso-Diaz and Laguna, 2006]. The idea of the statistical analysis is to provide a way to focus the local search on promising regions of the search space, helping to initialize the search at a non-arbitrary point. Calibra can only handle up to five parameters. The authors recommend to use the Taguchi's L_{16} (2^{15}) array to determine the five most significant parameters and fix the others to appropriate values, in the case that the algorithm being fine-tuned has more than five parameters. Using the Taguchi's array, Calibra can handle up to 15 parameters. Calibra works with integer arithmetic; however, parameters with continuous values must be discretized by specifying a level of accuracy (three decimal places).

- GGA: Gender based Genetic Algorithm (GGA) is a parallel genetic algorithm (GA) that uses genders for its individuals. The individuals are candidate configuration parameter [Ansótegui et al., 2009]. The authors say: mate choice is more likely to have a high impact on result quality than, for instance, natural selection. The GGA uses the concept of competitive and non-competitive genders, dividing all the individuals into two sub-populations with different genders (competitive and non-competitive). Different selection pressure on the two genders are applied. The algorithm allows the user to define the "design of the

[3]Iterated F-Race package available at `https://cran.r-project.org/web/packages/irace/index.html`

genome", i.e. relationship between parameters, through a tree. GGA can manage numerical and categorical parameters. The implementation makes use of parallel computations by taking advantage that a GA is easily parallelizable.

- HORA: It is a process proposed by Barbosa and Senne [2017], it is called Heuristic Oriented Racing Algorithm (HORA). The name comes from the use of a heuristic method and racing techniques. The idea of the authors is to consider the parameter configurations as a search space and to explore it for finding parameter values near to the promising (or the best) candidate configurations. The process starts with a number of instances selected arbitrarily from the given set of problem instances. The selected instances are treated as a training set. Experimental studies are carried out with the Response Surface Methodology (RSM) to define the best (promising) parameters settings for each instance. HORA executes a loop procedure, where dynamically creates new candidates parameters in the neighborhood of the promising candidate parameters and evaluates these with a racing method to discard poor ones (following statistical evidences using the Friedman statistic test). Through this procedure, HORA reaches better parameter settings iteration by iteration.

- SMAC: In [Hutter et al., 2011], the authors proposed two methods, Random Online Aggressive Racing (ROAR) and Sequential Model-Based Algorithm Configuration (SMAC). SMAC can be understood as an extension of ROAR. SMAC selects configurations based on a model rather than uniformly at random. SMAC also is an implementation of Sequential Model-Based Optimization (SMBO). The aim of SMAC is to remove some limitations that several SMBO methods (e.g., SPO) have. Limitations like only managing numerical parameters and tackling only one instance of a problem. To remove these limitations, the authors proposed: *(1)* a new intensification mechanism that employs blocked comparisons between configurations; *(2)* random forests as alternative class of response surface models, to handle categorical parameters and multiple instances; and *(3)* a new optimization procedure to select the most promising parameter configuration in a large mixed categorical/numerical space [4].

There is a variety of methods (or procedures or tools) to solve the PTP offline, all these with its specific features. Classifying them is a task that has already been done by other researchers. Dobslaw [2010b] distinguishes between two types of automated parameter tuning methods: *model-free* and *model-based*. The main difference between both two is that model-based approaches build a model. This model is created from the observations done by running the target algorithm with several parameter settings and interpreting the relation between the algorithm and its parameters. Model-free approaches do not create any model, it obtains implicit conclusions based on heuristic

[4]SMAC software available at `http://www.cs.ubc.ca/labs/beta/Projects/SMAC/`

rules; thus, the selection of interesting parameters is usually guided by randomness or by a simple experimental design [Dobslaw, 2010b].

Meanwhile, Huang et al. [2020] classify the methods into three categories: *simple generate-evaluate methods*, *iterative generate-evaluate methods*, and *high-level generate-evaluate methods*. The first are simple approaches, which consist of a generation phase and an evaluation phase. These are non-iterative methods that directly adopt this principle by firstly creating a number of parameter settings (candidate configurations), and then evaluating each of them for finding the best configuration. The iterative generate-evaluate methods involve a repeated process of generation and evaluation steps. These methods begin with a small set of initial configurations and create new configurations iteratively during execution. In this iterative execution, methods collect important information to make a smarter process and to choose better configurations. The last category, high-level generate-evaluate methods, has the main feature of quickly generate a set of elite or high-quality parameter configurations with a small need of computational resources; these methods carefully select the best configuration from this set instead of evaluating each candidate configuration thoroughly from the very beginning. In this category, the problem is to keep the diversity of elite candidates.

Table 3.1 summarizes the methods reviewed, proposed as PTS. The table shows the category if a method is model-based and its classification according to the generate-evaluate approach. Other important features are included in order to provide more information. For example, the table shows if the method uses some statistical test or some heuristic to guide the search. We also show if the method is implemented in parallel. The publication year and the programming language of the package (if there is one) are included too. The number of papers using each method is a difficult task to perform, so it is included the number of citations that have been made in google scholar until October 30, 2021.

Table 3.1: Summary of methods' features for PTS

Method	**Model based**	**Kind generate-evaluate**	**Statistical**	**Uses heuristics**	**Uses parallelism**	**Parameter Types**	**Year**	**Language**	**Google scholar citations**
DoE	✓	iterative	✓			all	19XX	R, python	9338
Revac		iterative				numerical	2006		121
ParamILS		iterative		✓		all	2009	Ruby	1005
EGO	✓	iterative	✓			all	1998		6424
SPO	✓	iterative	✓			numerical	2005	R	367
SKO	✓	iterative	✓			numerical	2006		692
F-Race	✓	simple	✓			all	2002	R	704
Iterated F-Race	✓	iterative	✓	✓		all	2007	R	285
Calibra	✓	iterative	✓	✓		numerical	2006		497
Brute-force		simple	✓			all	19XX		N/A
GGA		iterative		✓	✓	all	2009		348
HORA		iterative	✓	✓		all	2017		6
SMAC	✓	iterative		✓		all	2010	java, python	2029

The table 3.1 shows the characteristics of the Parameter Tuning Strategies. Most of these have been published in the last two decades, which shows the increasing importance of parameters setting in metaheuristics. Many of the methods are able to work

with all kinds of parameters (numerical and non-numerical). The main disadvantage of the method in general is that the amount of parameters are limited to a certain number. It is also possible to see that most of them have as a basis of their operation one or several characteristics, such as heuristics, some statistical analysis, iterative approaches or the creation of a model. Generally speaking, these methods are a time-consuming step for the designer. To speed this up, it is good practice to use parallel computing. From the state-of-the-art of PTS, only GGA uses this functionality.

3.2 Parameter Control Strategies (PCS)

According to the literature, online parameter setting or Parameter Control Strategies (PCS) is when the configuration of the parameters is accomplished as part of the algorithm execution while trying to solve the problem in question. Other less common terms found in the literature to refer to this approach are *Internal Parameter Control* [Fialho, 2010] or *Adaptive Metaheuristics* [Sevaux et al., 2015]. The latter definition also considers the modification of the whole behavior of the metaheuristic, changing the order in which the components are executed and/or the parameter values (i.e., the control flow of the algorithm).

For this kind of strategies, it is common to find that the adaptation mechanism is based on a behavior analysis of the metaheuristic during the search. The analysis implies getting some indicators (e.g., solution quality, similarity between solutions, local optima, among others). These indicators help to understand the behavior of the search. The knowledge gathered is used to adapt the parameters for obtaining better results during the process. The main drawback is the high amount of computational resource needed for solving the problem of the parameter and for solving the optimization problem. In addition, it is necessary to understand how the parameters influence the behavior of the method and how to identify which parameters are good for focussing the strategy so that prevail the best parameters [Calvet et al., 2016].

Modifying parameters during an algorithm execution in the context of Evolutionary Algorithms (EA) is not a new thing. The history of EA shows studies with this aim since the 60's and 70's [Fogel et al., 1966, De Jong, 1975, Schwefel, 1977], but at this time these strategies were known as evolution strategies. In EA, PCS has had the importance than other methods have hardly taken in the last two decades. Hence, various classifications can be found, and the latest and most complete taxonomy is the one proposed by Parpinelli et al. [2019]; this taxonomy combines the ones performed by Eiben et al. [1999, 2007] and Zhang et al. [2012]. The Parpinelli's taxonomy is not limited to EA but also focuses on another group of population algorithms, the swarm intelligence methods (SI). SI methods are characterised by algorithms inspired by the collective behaviour of insects such as ants, bees, termites and also animals such as fish and birds [Parpinelli et al., 2019], one example is the Ant Colony Optimization algorithm.

Parpinelli's paper reviewed 158 scientific articles, considering several aspects, such

as the control technique used, the domain of application, and the bio-inspired algorithm. They provide a broad overview about this topic, with the bonus that they consider both, EA and SI, areas that other reviews had not addressed. In this review, it is possible to note that there are algorithms with 2 parameters to 11. However, this applies not only for population methods. For single-solution methods, the number of parameters may also vary from few to many.

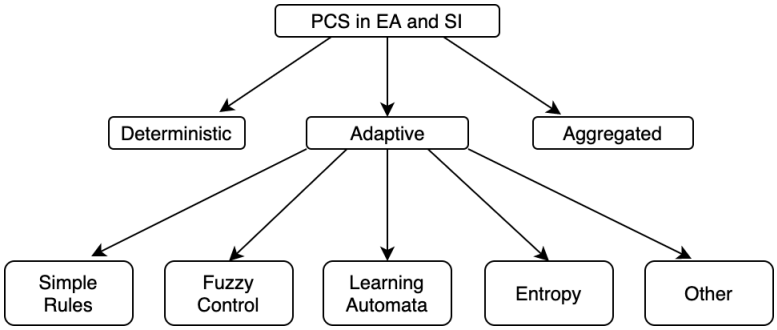

Figure 3.1: Taxonomy for parameter control in EA and SI algorithms, adapted from [Parpinelli et al., 2019]

Figure 3.1 shows the completed taxonomy for PCS in EA and SI algorithms adapted from Parpinelli's taxonomy. It is possible to identify three groups, *Deterministic*, *Adaptive* and *Aggregated* methods. The first group uses simple deterministic rules that modify the parameter values using no feedback during the execution. An usual action for a deterministic method is to initialize the parameter value with a big value and gradually decreases it during the optimization process. This action is done by trying to balance between diversification and intensification in the search. To perform the control, it is common to use the number of generations (iterations) or the number of fitness evaluations [Parpinelli et al., 2019]. Due to the simplicity and low computational effort, a lot of applications use this type of control.

A well-known strategy for Deterministic parameter control in EA is the 1/5th rule [Rechenberg, 1973]. This rule establishes that the proportion of success mutations for all mutations should be 1/5. Therefore, if the proportion is superior to 1/5, the mutation step size should be increased. If the proportion is inferior to 1/5, the mutation step size should be decreased [Parpinelli et al., 2019].

The second group presented in the taxonomy is *Adaptive*. This group is divided into five different subcategories: *Simple Rules*, *Fuzzy Control*, *Learning Automata*, *Entropy* and *Other*. Zhang et al. [2012] included *Aggregated* category within *Adaptive* category. The Adaptive control strategies consider a certain form of feedback from the algorithm. The feedback contains information of the optimization process which will be used to modify the parameters values. For EA and SI, the most commonly information used is the generation number, fitness evaluations number, and some population diversity measurement [Zhang et al., 2012]. The five subcategories are described next.

In *Simple Rules* methods, the control is carried out only using simple rules, based on the observations of the algorithm behaviour or runtime characteristics of the EA [Zhang et al., 2012]. Examples of these controls are the implementation of rules based on logarithmic, exponential, or linear functions [Fogarty, 1989].

In the *Fuzzy Control* methods, a controller converts an input (information from the feedback) to an output (actions for changing parameters) based on a set of fuzzy rules. These rules take inspiration on fuzzy logic. In [Herrera and Lozano, 2001] is summarized the fuzzy controller models applied to EA until 2001.

A *Learning Automata* method is used to select the parameter values by means of a learning process. The learning process is carried out according to the feedback obtained while performing the optimization. The aim is to update progressively its adaptation mechanism and the performance algorithm through the learning process. These methods learn from the previous experiences, being in permanent change, trying to refine its process and thus the results [Parpinelli et al., 2019]. It is possible to call a learning automata method as a machine learning technique.

The last classification subcategory named is *Entropy-based* methods. In the context of information theory, entropy is used to measure uncertainty of expected information, with a random variable. In the population-based algorithms, the generation of new population usually involves random and probabilistic operators. Thus, in this context, entropy can be used to analyze that population, getting characteristics and adjust the parameters accordingly. Shannon entropy is the most common control for entropy-based methods [Zhang et al., 2012]. Finally, the methods that not fall into any subcategory below are included in the *Other* methods, for instance, clustering, covariance matrix and pheromone matrix methods [Parpinelli et al., 2019].

The last category is *Aggregated* methods, also called self-adaptive methods [Eiben et al., 2007]. The main two features of aggregated methods are: *(1)* these contain the parameters directly coded with the solution vector, as an extra dimensions and *(2)* their parameters are adapted into the evolution process, using the same methods as the original algorithm (used for solving the optimization problem) or through specific routines [Zhang et al., 2012]. Zhang et al. consider self-adaptive control methods as a subset of *Adaptive*, because these methods are included in the adaptive definition.

A notable feature of the PCS methods for EA and SI is that they can be at the individual or population level. The individual level is when each individual (or solution) has its own set of parameters and the parameters are adapted specifically for this individual. On the population level, parameters are adapted for the whole population. This adaptation is done through fitness measurements of each individual or population-based indicators. An example of a population-based indicator is the relationship between the average and maximum fitness values of the population used in [Dai et al., 2006]. Population-based indicators are not restricted to population-based algorithms. For instance, it is possible to use them in parallel methods with single-solution metaheuristics (one or many) that have in their design a solution population.

In summary, we can conclude that, in the last decade, there have been proposed

several parameter setting methods for solving the PTP and PCS in EA and SI algorithms. For PCS methods in single-solution metaheuristics, the situation is quite different. If a researcher (or practitioner) is thinking of solving an optimization problem using metaheuristics with parameterization within the algorithm, there is no guide for classifying parameterization methods. Of course, there have been publications about this approach, however these have been done in isolation, specialized for a particular metaheuristic or a specific problem, not easily reproducible [Calvet et al., 2016].

The aforementioned problem has been partially solved by the Reactive Search Optimization (RSO) [Battiti and Brunato, 2010]. Battiti and Brunato define *reactive* word as follows: *"The word reactive hints at a ready response to events during the search through an internal online feedback loop for the self-tuning of critical parameters"*. This definition fits into the definition of Adaptive control strategies done for EA and SI algorithms, *"The Adaptive control strategies consider certain form of feedback of the algorithm, information of the optimisation process to modify the parameter values"* [Zhang et al., 2012]. However, the reaction possibilities of RSO consider not only to change the parameter values; these possibilities consider also changing the neighbors definition, the objective function, and the use of some constrains. Therefore, these all possibilities differ from modifying just the parameters as PCS. It is for reason that the RSO approach does not seem to fit completely with Adaptive control strategies.

Examples of PCS adjusted to the RSO approach are the Reactive Tabu Search (Reactive-TS) [Battiti and Tecchiolli, 1994] and the Adaptive Simulated Annealing (ASA) [Ingber, 1989]. For Reactive-TS, the tabu tenure, and for ASA, the temperature cooling schedule, are adjusted through feedback mechanisms during the search depending of the algorithm progress.

In the literature about PCS, it is possible to find other different approaches for parameter control in single-solution methods. These methods contain techniques like support vector machines [Zennaki and Ech-Cherif, 2010], probability matching techniques [Prais and Ribeiro, 2000, Neto and Martins, 2018], simple parameters exploration through parallelism [Blesa and Xhafa, 2004] and adaptation based on empirical analysis of the algorithm behavior [Ferro et al., 1996a]. However, some of the showed methodologies are not easily reproducible or are highly metaheuristic and problem dependent. These are some of the reasons why, in spite of the amount of parameter control works, many practitioners go on setting by hand or trying to design algorithms without parameters (or with a very low number of them), even with the proof that parameterizing an algorithm results in better algorithms leads to better performance [Calvet et al., 2016].

We can conclude that there is not method or methodology commonly accepted by the scientific community to setting the parameter of a metaheuristic. There is also a lack of methods (or research) for PCS and more for PCS in single-solution metaheuristics. The PCS for PSP is far from being solved, despite the methods mentioned above. However, it is worth mentioning and being aware that designing a PCS is much more complex than designing a PTS one [Dobslaw, 2010b]. This gap of PCS methods

makes this research work relevant in its theoretical and practical study. Additionaly, the importance of PCS as a vital part of the implementation and application of better metaheuristics methods, for the solution of optimization problems.

3.3 Instance-specific Parameter Tuning Strategies (IPTS)

The idea of Instance-specific Parameter Tuning Strategies (IPTS) is to find, for each specific instance of a problem, one static parameter values. PTS differs of IPTS, since PTS has the aiming to obtain one specific "robust" set of parameter values but for all instances of a problem. In IPTS each instance set is used to design an efficient tuning method that can return instance-specific parameter values based on measurable instance characteristics [Ries and Beullens, 2015].

In IPTS methods, the relation between the parameter values and the performance of the metaheuristic has to be strongly analyzed. This analysis is done taking into account instance specific features. IPTS avoids the need to modify the metaheuristic algorithm, reducing the possible computational effort required to adapt parameter values during algorithm execution as the PCS approach does. Another reason to implement IPTS is that looking for one fixed set of parameter values for an entire set of instances with different characteristics is not likely to perform well in all instances [Ries et al., 2012].

In simple words, IPTS works like this: a representative set of instances of the problem is selected, this set is investigated and used to design the tuning method. Once designed, the tuning method calculates the characteristics of any given instance, and then applies a method (or model, function) to return a custom parameter values for a specific instance. These values are used to initialize the metaheuristic for solving the specific instance, and these parameter values remain fixed during the execution [Ries and Beullens, 2015]. Ries and Beullens noted that an IPTS design has several challenges, like the selection of relevant characteristics, as well as the identification of the relation between: *(1)* the instance-specific information, *(2)* the parameter values of the algorithm, and *(3)* their impact on running times and quality of solutions obtained.

Is possible to think IPTS as a subcategory of PTS, firstly, because the IPTS name suggests it and second by the fact that the parameters for solving the problem instance remain fixed during the execution, even if they are different for all instances. Nevertheless, there is in the literature another definition that can make a difference between IPTS and PTS. Some authors define and have used the IPTS as the combination of the advantages of PTS and PCS [Ries and Beullens, 2015, Calvet et al., 2016, Dobslaw, 2010a].

Only few approaches can be currently found in the literature that used IPTS or that may fall within the definition of this approach. As we have said, the differences in their definitions are still a problem to be solved for this area of research. For

that reason, some papers have been published as PTS approach, for example [Pavón et al., 2009, Dobslaw, 2010a]. The Dobslaw's work is interesting, it combines the advantages of Design of Experiments (PTS) and Artificial Neural Networks (PCS) for the recognition of good initial parameter settings for new problem instances. They tested their methodology adjusting the parameters of the Particle Swarm Optimization algorithm.

Ries et al. [2012] introduced the concept of IPTS in 2012, even they mention *instance-specific parameter control strategies (IPCS)*, in order to adapt the dynamic parameter control procedures to the specific characteristics of each case. In view of the state-of-the-art of PSP, IPCS is an unused approach.

As the IPTS studies has shown, an important part is to carry out a statistical analysis to identify the impact of parameters on the metaheuristic performance and to know the interaction between them. However, the statistical analysis requires someone to interpret the results. The most of IPTS methods have been designed by using this statistical analysis and all the knowledge of the problem gathered [Ries et al., 2012, Ries and Beullens, 2015, Dobslaw, 2010a]. In [Ries et al., 2012], they proposed a fuzzy logic method based on the knowledge obtained from the statistical analysis. The fuzzy logic method is a rule-based approach, where the control objectives and relationships between inputs and actions (or outputs) are captured in the fuzzy logic system, usually through a set of IF–THEN rules.

Ries and Beullens [2015] proposed a semi-automated approach for designing fuzzy logic, whereby the classification rules are derived from *automatically* generated decision trees. With this automation, they has the intention of removing the requirement of having an expert to interpret of the statistical results. According to the authors, the mentioned approach is generally applicable for developing an IPTS for any metaheuristic and type of problem. They tested it to parameterize a Guided Local Search for solving the Travelling Salesman Problem.

In previous work, Pavón et al. [2009] propose the idea of having methods without the utilization of statistical analysis. They designed a hybrid system, using Case-Based Reasoning (CBR) and Bayesian Networks (BN). BNs are used to model qualitative and quantitative relationships between the different algorithm parameters, while the CBR methodology is used to update the models associated with each case and to learn new cases within the domain. The system was done to tune a Genetic Algorithm for solving the root identification problem. This system is an extension of the model done in 2008 [Pavón et al., 2008]. This last also uses a BN to capture the dependencies between the parameters and the performance of the algorithm.

It is gaining attention to consider the instance-specific information. Knowing this information, and learning how to interpret it, can be crucial to achieve better methods and better results in less computational time. For the time being, the number of IPTS works are low, since it is relatively new approach. So, proposing a taxonomy does not make sense now, however, this chapter shows the highlights of IPTS.

3.4 HyperHeuristics (HH)

The term hyperheuristic was first used in a conference paper [Cowling et al., 2001]. A single pre-2001 occurrence can also be found in a technical report where it was used in a different context, to describe an approach that combines a range of artificial intelligence algorithms for automated theorem testing [Denzinger et al., 1996]. However, the basic idea of automating the design and/or selection of metaheuristics with its **parameters** is much older. It dates back to the early 1960s [Burke et al., 2013], as mentioned also for the Evolutionary Algorithms [Fogel et al., 1966, De Jong, 1975, Schwefel, 1977].

Related to this emerged new concept, several publications have attempted to classify the work that has been done [Burke et al., 2003, Ross, 2005, Chakhlevitch and Cowling, 2008, Burke et al., 2009, 2013, 2019]. Burke et al. [2009] discusses methodologies for generating new metaheuristics from a set of potential metaheuristic components, in which genetic programming (GP) plays a prominent role as machine learning technique and as high-level heuristic. Later, Burke et al. [2013, 2019] propose a unified classification and definition that captures all the work being done in the field of hyperheuristics, showing that it is a promising research area.

Thus, since the introduction of the hyperheuristic concept, there have been important advances in solving combinatorial optimization problems. The design of an *Evolutionary HyperHeuristic* with the implementation of evolutionary algorithms [Oltean and Grosan, 2004, Guogis and Misevičius, 2014], such as genetic programming (GP), gene expression programming (GEP), and grammatical evolution (GE), acting as a high-level heuristic has been successfully applied [Su et al., 2011, Dokeroglu and Cosar, 2016, Garrido and Riff, 2010, Marshall et al., 2014, Sabar et al., 2013, 2014, 2015]. Non-evolutionary approaches such as Roulette wheel, Monte Carlo algorithm, and others have also been used as high-level heuristics with satisfactory results [Marshall et al., 2014, Sabar and Kendall, 2014, Sim and Hart, 2014, Turky et al., 2018].

Due to the growing interest in hyperheuristics, in 2011, during the first Cross-Domain Heuristic Search Challenge Competition (CHeSC), HyFlex, a flexible framework for the creation of hyperheuristics, was presented [Ochoa et al., 2012]. HyFlex provides six difficult combinatorial optimization problems, is public and is implemented in Java. Multiple applications have been done using HyFlex [Marshall et al., 2014, Sabar et al., 2014, Sabar and Kendall, 2014, Sabar et al., 2015, Su et al., 2011].

It can find in the literature that just one hyperheuristic work solves the QAP and uses parallelism in its design. In this study, the authors proposed a high-performance Multistart Hyper-heuristic Algorithm (MSH-QAP). MSH-QAP makes use of some of the metaheuristics that have been reported to be among the best performance algorithms for solving difficult QAP instances, Simulated Annealing, Robust Tabu Search, Fast Ant System, and Breakout Local Search. The high-level heuristic is performed by a genetic algorithm (GA) and is called master agent hiperheuristic. MSH-QAP has two execution phases [Dokeroglu and Cosar, 2016]. This method is a *Evolutionary HyperHeuristic*, because it has GA as high-level heuristic.

In the first phase of MSH-QAP, in the GA each individual in the population represents as a whole a solution, a metaheuristic method and its parameters. In each generation of the execution by parallel computing, each available execution unit processes a metaheuristic with its parameters and returns its result to the master agent hiperheuristic. The master agent generation by generation performs the **adaptation of parameters** through crossing and mutation operators (similar behavior to *Aggregated* category in PCS methods). At the end the parameters with their best adjustment for each method remain. If in this first phase the best known solution is not reached, phase two is activated. In phase two, the best metaheuristic is selected with its parameter settings, it is given to all execution units and executed with multiple starts.

It can be seen that the lines that differentiate some *Evolutionary HyperHeuristic* with the *Aggregated* methods in PCS are diffuse. That is, there are works that can be classified by its design as *Evolutionary HyperHeuristic* as *Aggregated*, for instance the MSH-QAP method.

3.5 Taxonomy PSP Methods

Figure 3.2 depicts our complete taxonomy about PSP, it summarizes all the information reviewed here. As we mentioned earlier, the literature about PSP is highly different and varied. In the last two decades, this problem has been of great interest to researchers and practitioners who study optimization problems and metaheuristic methods.

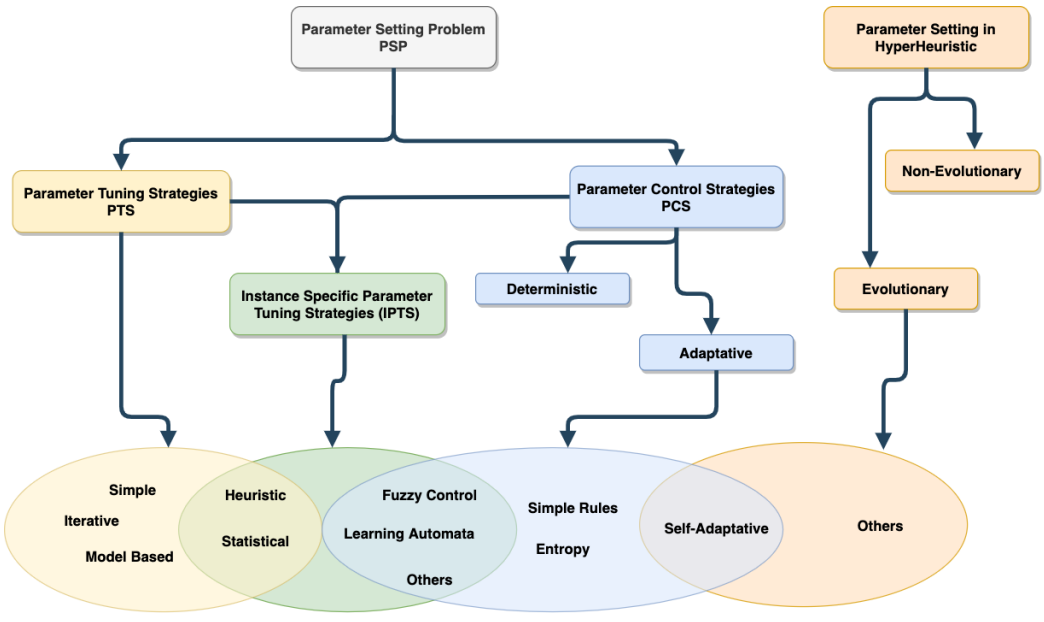

Figure 3.2: Taxonomy of strategies to solve the parameter setting problem (PSP)

The task of classifying all methods to solve the PSP had not been done until now. Here, we propose a taxonomy that generalize the taxonomies made individually for each category of PSP (PTS, PCS, IPTS and HH). In this taxonomy, it is possible to identify that there are an overlapping of subcategories, which means there are subcategories that are part of two categories. This suggests two things, the first is the definition of a subcategory is sufficiently broad or ambiguous to fit into two categories or that more studies are needed to show more clearly their differences. We consider the latter to be the main reason, due to the newness of the PSP methods and the research gaps that still exist in many of them (e.g. PCS methods for single solution metaheuristic).

To make this taxonomy, we classified PTS methods in the next subcategories: *Simple*, *Iterative*, *Heuristic*, *Statistical* and *Model-based*. This classification is done based on the characteristics we analyzed in the summary table 3.1 of the section 3.1. Thus, those methods that use simple strategies to select the parameters are *Simple*, those who use an iterative process to perform the selection are *Iterative*, those that use some heuristic to guide the search to select good parameters are *Heuristic*, those who use some statistical test are *Statistical* and those who build a model are *Model-based*.

From the figure 3.2, it is possible appreciate that the methods classified as *Heuristic*, *Statistical*, *Fuzzy Control*, *Learning Automata*, *Self-Adaptative* and *Others* also belong within two big superior categories, for example *Heuristic* belongs to PTS and IPTS.

In the case of those methods that belong to IPTS category, all belong either to PTS or PCS, a situation that was expected, since as mentioned IPTS methods contemplate the combination of PTS and PCS. Finally, *Self-Adaptative* methods (or *Aggregated* in PCS for some authors) belong to both PCS and parameter setting in *Evolutionary Hyperheuristics*. This is because *Self-Adaptative* methods manage the parameters together with the solution of the problem within the evolutionary process [Zhang et al., 2012].

We can conclude that knowing which method is best for solving the PSP is a difficult task to assess, in general each one offer different advantages and have many disadvantages. The dynamic adaptation of the parameter values that characterizes PCS usually provides better results. However, the computational effort tends to be higher. On the other hand, the PTS approach is the easiest and fastest to use. Once a set of parameter values is selected, the algorithm code does not have to change to find the set of parameter. Nevertheless, finding an adequate set may be also time-consuming. Finally, the IPTS group of strategies represents a compromise strategy: it takes less computational time than the PCS approach, but requires implementing a learning mechanism to interpret the relation between the relevant metaheuristic characteristics selected [Calvet et al., 2016].

Therefore, there is no approach that stands out from the others. Probably, the most adequate depends on the specific problem to tackle, the instances to solve, the available time and the skills of the researcher. Despite this fact, some general guidelines can

be formulated. PTS can be considered as the best option when working with robust algorithms [Calvet et al., 2016] and when the instances to be solved are somehow similar in their characteristics due to the specific area of application in which they occur [Ries et al., 2012]. Regarding IPTS, they are more complex than PTS, but provide better results when the algorithm is not robust and at the same time, when the instances differ in their characteristics.

In case of prioritizing the algorithm performance, PCS usually constitute the most recommendable approach [Calvet et al., 2016], these strategies work arguably best when the instances that need solving are not a priori well known but are expected to differ significantly in their characteristics [Ries et al., 2012]. Finally, in order to create a general method for solving various optimization problems, HH is the best choice.

Chapter 4

Framework

4.1 Introduction

As we pointed out in the state of the art, there is a gap in parameter control strategies for single-solution metaheuristics. The research problem is even more notorious in parallel hybrid metaheuristics, since these methods need to fine tune a large number of parameters (more metaheuristic of different types are involved and there are parameters to define their parallel hybrid behaviour).

In this chapter, we propose PACAS: a general framework to configure the **PA**rameter **C**ontrol **A**daptation for **S**ingle solution metaheuristics in a parallel hybrid solver. The PACAS framework aimed at increasing the performance of parallel method based on several parameter control strategies through the analysis of the behavior of single solution metaheuristics. This framework allows the programmer to define the adaptation of the parameter using some mechanisms to customize the trade-off between intensification and diversification in the search process.

In the remaining of this chapter we describe the general concepts of our PACAS framework (Section 4.2), the subsections within this section contain details of the team structure and the description of the functionality of each team component. In addition, the mechanisms for setting the parameter control strategies are exposed in Section 4.3. Then, we present a summary of the framework parameters (Section 4.4). Finally, we introduce a dedicated section to discuss the sources of hybrid behavior within the framework.

4.2 General description

In this section, we describe the design details of our framework, PACAS. Searcher nodes are the basic components of the framework: each searcher node corresponds to a single solution metaheuristic solver instance of any type. The idea is to set each searcher node ideally bound to its own dedicated core, using all available processing hardware. One of the basic uses of the framework is for running several instances of

different metaheuristic, in an independent parallel search with static parameters, all of them starting from a different search point, i.e. different initial solution.

The framework allows cooperation through a solution population and allows change dynamically the setting parameters of each single metaheuristic every certain time. In PACAS is possible to control when accept or how select a solution from the solution population by means of a request and entrance policy. The adjustment of the request and entrance policy and the mechanisms defined for the adaptation of the parameters contribute to the strategies to balance diversification and intensification. Even the configuration of these policies, and the moments related with the application of them, i.e. the time associated to set how often this task is done, establish the hybrid behavior and the amount of cooperation among the searchers. A master node manages these policies, adaptation mechanisms and times. The master node additionally carries out a performance evaluation of each searcher node using some collected information in order to apply the parameter adaptation mechanisms.

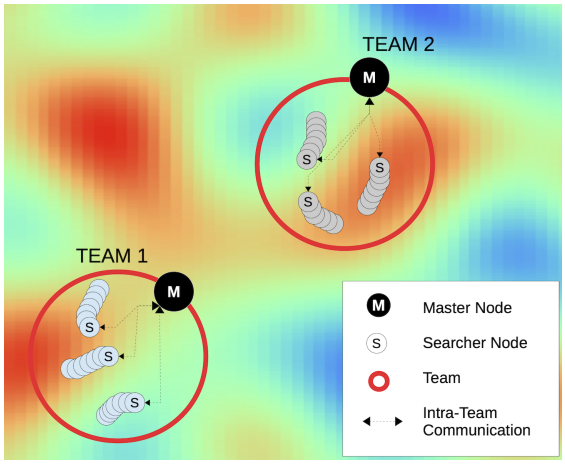

Figure 4.1: Representation of the search process using PACAS

Searcher nodes can be grouped into teams of fixed size, teams with the same number of metaheuristics of each type. For example, teams configured to have 10 Local Search, 10 Tabu Search and 10 Simulated Annealing. Thus, if we have 2 teams, it means 60 metaheuristic in total, 20 for each type. Each team inside has its own request and entrance policy. For instance, one team could have an elite entrance policy, to control that only the best solutions can enter in the solution population, and a second team implemented a random entrance policy, where a solution comes out randomly to give space for a new solution. Within a team, it would also be possible to design different versions of the mechanisms used to make the adaptation of the parameter (feature will be explained in detail later, Section 4.3). There is no communication between the teams, so it is important to be careful with the policies and mechanisms defined, the definition of these both features work together for having a good trade-off between intensification and diversification. On the head of each team there is a master node.

Figure 4.1 depicts the search process using PACAS. Among the master node and the searcher nodes, there is communication (Intra-team communication). The Intra-team communication is performed synchronously by each searcher node every certain time. The communication process is necessary for the master node to obtain the behavior of the metaheuristics, make the performance evaluation and later apply the establish parameter control strategies (PCS). The PCS involve two stages, (1) evaluating the searcher node behavior and (2) adapting the parameters. For each stage some rules and mechanisms must be defined.

4.2.1 Team structure

In our framework, a team is integrated by one master node, several searcher nodes and one solution population (SP). All these components work together in an environment controlled by the master (see Figure 4.2). Our framework proposes an intra-team communication as a cooperative mechanism, to ensure inside this environment the balance of intensification and diversification. Each team is a parallel cooperative search.

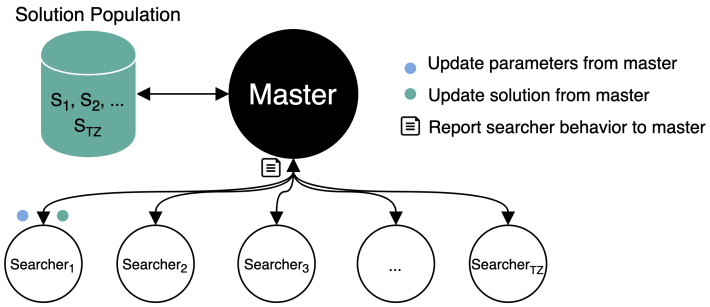

Figure 4.2: Structure of a Team

Each searcher node reports periodically its current candidate solution and some contextual information (e.g., solution cost, performance metrics, parameters, etc. See Figure 4.3) to the master node, which stores (or not) intermediate solutions into the SP based on the input policy specified for the team. This mechanism promotes diversity (much or few) for the candidate solutions in the pool, according to its definition. This process constitutes also a flexible interaction feature which eases the hybridization of metaheuristics, promoting cross-fertilization among different types. The size of SP is equal to the number of searcher nodes per team (team size, `TZ`), since the population just has a solution for each searcher node. Therefore, one of the first things of the framework to be defined is the team size (`TZ`) and the number of teams (`TN`). The main parameters associated with these features are:

- Team Size, `TZ`: is the number of searcher nodes per team. All teams have the same `TZ`. `TZ` is also the size of the SP.

- Number of Teams, `TN`.

`TZ` and `TN` are linked to the maximum number of available cores for the execution, the multiplication of `TZ` and `TN` results in the total number of cores to be used. These two parameters are directly related to the trade-off between intensification and diversification. Depending on the number of cores available for execution, a large number for `TN` may mean a small number of searchers, `TZ` (i.e. small SP size) which would be expected to give a high level of intensification. On the other hand, a small number of teams (1 or 2 are a good choices) means a large number of searcher nodes for the team (i.e. a large SP size) and the framework would have a very high level of diversification. However, the main configuration for intensification and diversification is given by the PCS inside each team.

The master node is on the top of the team, who implements a parameter control strategy (dynamic adaptation) which is tasked to automatically adjust the parameters of the searcher nodes during their execution. Master node receives and processes all the information from the searcher nodes, by means of a searcher report. Master has a global vision of what is happening in the team and make decisions based on the comparative performances of the searcher nodes.

4.2.2 Master node

The master node is the team leader. It is in charge of guiding each metaheuristic method to reach its best parameter settings trying to strike a balance between intensification and diversification in the search. The master operates within an iterative process. At each iteration, each metaheuristic runs for a given time, iteration time. When the iteration time is running out, searcher nodes report to the master node the initial solution, and the current solution (best found) in the interval with theirs associated costs and parameters, with other identification searcher data, all these information is encapsulated in a report (figure 4.3).

When master receives this report, it develops a *performance evaluation* for each searcher and executes the *parameters' adaptation* procedure. The master then sends a new evolved set of parameters and a new configuration back to the searcher nodes. Searcher nodes resume the search with the settings they received: parameters and restarting from a new initial solution for the next iteration. PACAS repeats this process for each team until an established number of iterations are accomplished or when the solution target is reached. The number of iterations indicates the number of times the master node carries out the *performance evaluation* for each searcher and executes the procedure for adapting its parameters, i.e. the total number of parameter adaptations. Thus, related to the execution times, the number of iterations or total adaptations (`A`) and the iteration time (`I`) must be defined by each team.

- Total adaptations, `A`: it defines the total number of times the searchers communicate with the master and the number of times the master executes the PCS.

- Iteration time, `I`: it determines how often a searcher communicates with the master node and the time that a searcher runs an instance of the metaheuristic searching for the solution of the optimization problem.

Thus, the total execution time of the algorithm is given by the product between the total adaptations and iteration time parameters (`A*I`). For instance, if 15 adaptations (`A`=15) are performed and each searcher node runs for 20 seconds (`I`=20), the total execution time would be 5 minutes (15*20 = 300 seconds). This product must the same for each `A` and `I` value of each team. But the values of `A` and `I` of one team do not have to be equal to those of another.

For the cooperation within the team, the master manages a SP, this population contains the solutions reported by the searchers that were able to enter based on the input policy. The SP has the next configuration parameters that must be defined at design time:

- Entry Policy, `EP`: it defines the rules to accept or reject incoming solutions.
- Request Policy, `RP`: it defines how to select the solution which is actually delivered to a searcher node when it makes a request, then searcher use that for running during the `I` time.

For example, a possible entry policy: when a report from the searcher node is received by the master node, the master simply ignores the incoming solution (instead of storing it in the SP), in the following situations:

1. If the solution cost is worse than the worst cost in the SP (i.e. elite policy).
2. If the solution is already stored in the SP (in this case, it can be mutated by performing two random swaps)

There are several choices for the request policy in the master node when sending a solution to a searcher. A simple and effective strategy is to return a random solution from the SP. But other alternatives exist, for instance return the best solution in the SP or the same solution that the searcher node reported (producing a non-cooperative behavior in the team).

4.2.3 Searcher nodes

Searcher nodes run a metaheuristic single-solution instance, in parallel, carrying out the search process. They periodically send a report to the master node every `I` time (iteration time). This report contains the data showed in figure 4.3. The master processes the report, executes the `EP` policy to SP for the current solution (which correspond to the best found in the interval) and applies the parameter control strategies for the current parameters. The searchers waits while the master node processes the

report, when master finished, it sends to the searches a new, adapted, set of parameters and a new solution depending the request policy (RP). Then, the searcher nodes resume the search during the I time and sends a new report with the updated behavior data to the master. This process is repeated until complete an established number of total adaptations (A) or when the solution target is reached for any searcher node in the whole algorithm.

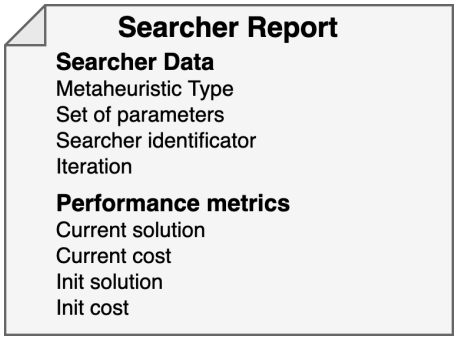

Figure 4.3: Searcher report

4.3 Parameter Control Strategies

The parameter control strategies (PCS) provide mechanisms to intensify and diversify the search. The master executes these strategies. In order to know how to properly modify the parameters, an analysis of the execution behavior of each metaheuristic during its I time is done. This analysis is based on the search history sent to the master through the report.

Evaluating the performance of the metaheuristics is a critical process, and selecting the right set of metrics affects the overall performance of the *parameters' adaptation process*. In PACAS, this *performance evaluation* is based on two indicators, **percentage gain** in the cost of the objective function and **distance between solutions** (pair-wise difference). The gain acts as a direct indicator of the metaheuristic's performance, meanwhile the distance is assessing how diverse the search is. To adapt the parameters, these indicators will be compared against the diversification limits (DL) and the solution similarity percentage (SSP) respectively. So, for applying these strategies to each team, it is necessary to define:

- Diversification Limits, DL: it defines the limit values of cost percentage of the objective function, values to be compared with the percentage gain and used to define the directive sent to PCS to intensify or diversify.
- Solution Similarity Percentage, SSP: it defines similarity percentage value when two solution are very similar. Value to be compared with the pair-wise difference and used to define the directive sent to PCS to intensify or diversify.

When the indicators are lower than the corresponding values for DL and SSP, the framework executes a PCS strategy to promote diversification in the search. When the percentage gain is higher than the corresponding DL or the pair-wise difference is higher than the SSP, the framework adapts the parameters to promote intensification in the search.

Based on the evaluation of the searcher's performance, the framework produces a directive which can be, intensify or diversify. This directive is used as input for the parameters adaptation process, PCS. For each directive case, it is necessary define a behavior depending on the metaheuristic type implemented.

Considering that a metaheuristic may have different type of parameters, numerical and categorical (or non-numerical), we define different strategies for adapting these parameters. If the parameter is numerical, it is adjusted by adding deltas to their current values. If the parameter is categorical, it is changed depending on a number of iterations without improvement in the cost of the objective function. Therefore, for each parameter of each type of metaheuristic, it is necessary to define:

- Delta to diversify, DF: it specifies the value to be added to the numerical parameter to promote diversification.
- Delta to intensify, DI: it specifies the value to be added to the numerical parameter to promote intensification.
- Iterations without improvement, IWI: it determines the number of iterations when a categorical parameter has to be change to another option different to the current one.

To establish DF, DI and IWI, it is necessary to understand how the parameters influence the behavior of the method, to know which parameters are good for intensification or diversification and thus focus the strategy actions so that the best parameters prevail or adapt. PACAS also allows to adapt the parameters randomly or to keep them constant during the whole execution.

4.4 Framework Parameters Summary

Table 4.1 summarizes of the complete set of parameters for the PACAS framework. The first and second columns contain respectively the short names and full names of the parameters. The last column briefly describes them.

4.5 Hybridization in the PACAS framework

In this section, we explain in more detail those features of PACAS to provide a hybrid algorithm behavior. Our proposed PACAS framework provides different sources of

Table 4.1: Summary PACAS framework parameters.

Short name	Parameter name	Description
TZ	Team size	Number of searcher nodes per team
TN	Number of Teams	Total number of teams of the whole framework
A	Total adaptations	Total parameter adaptations to be done by a master node
I	Iteration time	Time for a searcher runs a metaheuristic instance
EP	Entry Policy	Rules to accept or reject incoming solutions into SP
RP	Request Policy	Rules to select a solution from SP to be delivered to a searcher
DL	Diversification Limits	Cost limit values where the parameters are adapted to diversify
SSP	Solution Similarity Percentage	Similarity where the parameters are adapted to diversify
DF	Delta to diversify	Value to be added to the numerical parameter to diversify
DI	Delta to intensify	Value to be added to the numerical parameter to intensify
IWI	Iterations without improvement	Iterations number when a categorical parameter has to be change

hybridization, by design of the framework. Although PACAS does not define dedicated modules to implement hybridization in the framework, it provides many attributes that can be classified as parallel hybrid metaheuristics. The sources of hybridization are affected according to the user configuration.

The first source of hybridization is associated with the **intra-team communication** mechanism. In this case, searchers may implement different types of metaheuristics methods which interact through the exchanged solutions in the master node's solutions population (SP). This is clearly an example of high-level hybridization [Talbi, 2002], because the methods are self-contained in the searcher without a significant modification of their functionality. Thus, the hybridization behavior in a team depends on how the PACAS framework is parametrized, specifically the `EP`, `RP` and `I` parameters. These parameters define how to process solutions in the SP, and how often this task is done. For instance, if the update interval time (`I` parameter) is set to a low value, the metaheuristics frequently changes its solution to another solution, thus the framework promotes a frequent interaction for cooperation i.e., more opportunities for hybridization. Otherwise, if the update interval is large, the metaheuristics are executed with a given solution for a long period, that means low frequency of cooperation and hybridization, and the metaheuristics have enough time to do a large run. In this case, an undesired behavior can happen, such as getting stagnation in a local optimum, if the methods do not have strategies to escape from them.

The second source of hybridization is granted by the use of **different types of metaheuristics**. Combining in a parallel execution diverse methods provides benefits from the individual advantages of each one. In PACAS, each searcher is encapsulated without a significant modification of their functionality. Hence, each method, being characterized by different strengths and weaknesses, will explore different regions of the search space. All this search enrichment materialized by the solutions reported to the SP, will be distributed by the master node to the searchers depending on the `RP`. The entire system behaves as a hybrid solver, benefiting from cross-fertilization which

derives from the inherent diversity of the search methods strategies. These two sources of hybridization work together to yield a high or low amount of hybridization, which will depend on the configuration of the `EP`, `RP` and `I` parameters and the number of different metaheuristics implemented.

Chapter 5

A Java Implementation of PACAS framework

This chapter presents the detailed description of our Java implementation of the PACAS framework, J-PACAS. This implementation is an essential part of the design cycle of the framework, allowing us to validate all features of it.

5.1 Introduction

In the previous chapter, we specified all the features of the PACAS framework. The implementation of the framework is an important component of the design process, which allows to evaluate its behavior. The selection of a suitable programming language is an important decision that is necessary to make first. This decision directly affects the functional requirements and non functional requirements. We decided to use *Java* based on: working community, performance, development complexity, standards for parallel programming and programming models supported. Although *C* supports all these features, the complexity of the language makes the development process error-prone and slow. We took into account this and the fact that general and optimization working community is increasingly using Java for their projects, for example: SMAC [Hutter et al., 2011], HyFlex [Ochoa et al., 2012]. We considered other languages such as *C++*, *Scala*, *Python*, *Rust*, *Erlang* and *X10*.

Java is a general-purpose, concurrent, strongly typed, class-based object-oriented language. Thanks to the Java Garbage Collector, thorough type checking and the absence of pointers make Java development significantly fewer error prone than more traditional languages such as *C*, *C++* and *Fortran*. One of our concerns was the performance and portability. However, Java compilers have made rapid advances in this field and its portability makes it possible to run a project on almost any architecture. Another important criterium is the support for parallel programming. Here, Java has support for distributed and shared memory models. All these features become Java a suitable language to develop a framework for solving optimization problems.

Our J-PACAS framework is implemented in Java 11 using the `ForkJoinPool` and `AtomicType` classes to handle the parallelism in a shared memory model.

The rest of this chapter presents the implementation details of the J-PACAS, providing guidelines to extend the solver functionality and to configure all its parameters. Section 5.3 concludes this chapter.

5.2 Implementation details

J-PACAS is our implementation in the *Java* programming language of the framework **PA**rameter **C**ontrol **A**daptation for **S**ingle solution metaheuristics (PACAS), which is available in open source code through a git repository [1]. Figure 5.2 shows the main classes of J-PACAS and shows how they are related (Figure 5.1 shows relation notation). Some of them correspond to **specific** classes (like *ParameterControl*, *TeamConfiguration*, *SolutionPopulation*, among others) and others are **generic** classes (like *GenericTeam*, *MetaheuricticSearch*, etc). Generic classes are used as a base to extend the functionalities of the framework through the inheritance and polymorphism mechanisms of object-oriented programming.

There is a generic metaheuristic class (*MetaheuristicSearch*), a generic team class (*GenericTeam*) and a generic problem class (*GenericProblem*). The functionality of the PACAS, like the team, the searcher node, and the master node, are contained in the *GenericTeam* class. Solution Population (SP) and methods for interacting with it are implemented in *SolutionPopulation* class. The *GenericTeam* has one *SolutionPopulation*, it manages the entry and request policies. The PCS are established in the *ParameterControl* class and the complete configuration for an execution of the J-PACAS framework is defined in the class called *AlgorithmConfiguration*. In the following subsections, we detail the implementation and the role of the main classes in J-PACAS.

Notation	Meaning
———	Relationship
——‖	One and ONLY One
——⋉	One or Many
——○⋉	Zero or Many

Figure 5.1: J-PACAS Class Diagram Notation

[1] source code and instances are at `https://github.com/JonathanDuque/QAPMetaheuristic/tree/framework`

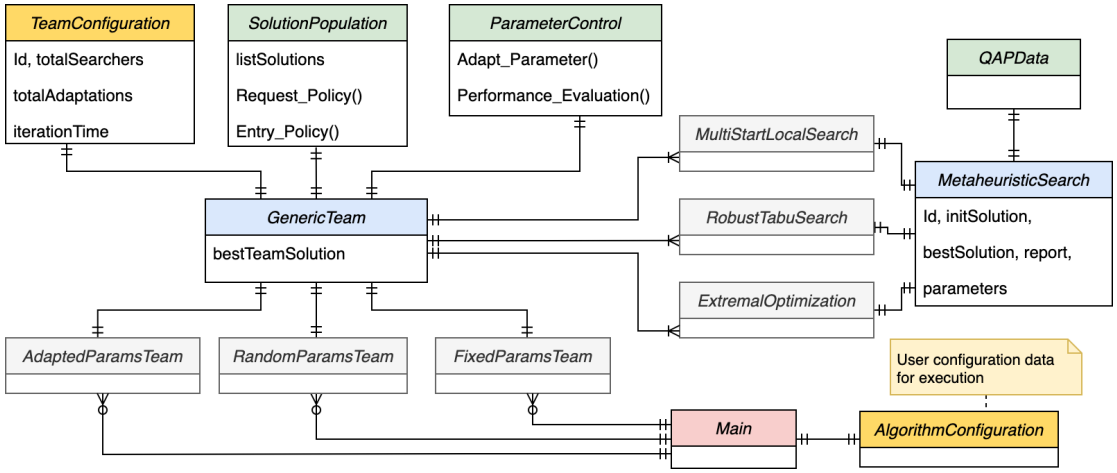

Figure 5.2: J-PACAS Class Diagram

5.2.1 MetaheuristicSearch Class

The *MetaheuristicSearch* class defines a base search method, which implements the necessary functions and variables to interact with the components of the PACAS framework to perform a search. Every searcher in the J-PACAS implementation is a subclass of the *MetaheuristicSearch* class. To implement a new searcher node, the programmer must create a new search class that inherits its functionality from the *MetaheuristicSearch* parent class. In this a new searcher, the programmer must overwrite just one method that defines the searcher specific functionality, the *compute* method. Table 5.1 presents a summary of the main methods of *MetaheuristicSearch* class, along with a brief description of their functionalities.

Table 5.1: Main methods in the *MetaheuristicSearch* Class

Method name	**Description**
Getters methods	These are the methods to get: initSolution, bestSolution, parameters, metaheuristicReport, threadLocalRandom, among other useful data.
Setters methods	These are the methods to set: bestCost, and bestSolution data.
void setEnvironment(int[] , int[] , int)	Function to set the environment initial data for a searcher node. Method should be called once before the main loop of the algorithm.
void compute()	Implements the functionality of a single iteration of the search process in an iteration time (`I`).

Inside the *compute* method of the *MetaheuristicSearch* class the searcher performs the search to get the solution of the problem during the iteration time. To overwrite this method, user has to do the following steps: (1) create a loop that will be executed for an iteration time or until find the target cost (normally the BKS), (2) set the bestSolution and the bestCost. For more details, see as an example the implementation of the MLS metaheuristic, *MultiStartLocalSearch* class.

J-PACAS provides three different metaheuristic, that can be utilized as a searcher node: Robust Tabu Search (RoTS) [Taillard, 1991], Extremal Optimization (EO) [Munera et al., 2016], and Multistart Local Search (MLS). Each netaheuristic implemented includes its deltas of each parameter for adaptation. These metaheuristics are commonly used in combinatorial optimization problems, particularly for the QAP. We now present a brief description of each of these methods and their parameter ranges.

Robust Tabu Search: The name Tabu Search (TS) refers to the use of an adaptive memory and special problem-solving strategies, called intelligent strategies, in order to get a better local search method [Glover, 1990]. The idea is to memorize within a structure the elements that for the LS will be forbidden to use (*tabu*) and thus avoid getting trapped in local optima. TS looks for the best solution within the neighborhood but does not visit the solutions of previous neighbors if they have been visited before or have been marked as prohibited locations [Dokeroglu and Cosar, 2016]. Robust Tabu Search is an adaptation of TS to the QAP and has been one of the best performing methods for this problem [Taillard, 1991].

Extremal Optimization: Extremal Optimization (EO) is a metaheuristic inspired by self-organizing processes as frequently found in nature: for EO this is *self-organized criticality* (SOC). The version proposed by [Boettcher and Percus, 2000] has only one adjustable parameter: τ, and uses of a Probability Distribution Function (PDF). EO proceeds like this: it inspects the all candidate configurations (all assignments in the neighborhood). Each one is assigned a fitness value by means of the goal function. The configurations are then ranked from worst to best. After that, the PDF introduces uncertainty in the search process. EO resorts to the PDF to choose a solution from organized configurations. The role of the τ parameter is to provide different search strategies from pure random walk ($\tau = 0$) to deterministic (greedy) search ($\tau \Rightarrow \infty$). In a recent work, the basic EO metaheuristic was extended to support not only a power-law PDF but also an exponential and a gamma-law PDFs [Munera et al., 2016].

Multistart Local Search: A simple Local Search (LS) is one of the oldest and most frequently used metaheuristics. LS starts from an initial solution and repeatedly improves it within a defined neighborhood. Neighbor solutions can be generated by applying minor changes to the initial solution. LS ends when no improved solutions are found in the neighborhood of the current solution, achieving a local optimum [Yagiura and Ibaraki, 2002]. Multistart Local Search (MLS) is a modification of LS that iteratively performs multiple different searches, executing each local search from a different starting point. When MLS reaches a local optimum, it tries to escape by restarting the search from scratch or performing some random moves in the current solution. This metaheuristic is also known as Iterated Local Search [Lourenço et al., 2003].

Parameter ranges of the implemented metaheuristics: Table 5.2 presents the parameters considered for each metaheuristic implemented, together with the range of variation for each parameter. These ranges are picked from the best performances,

as reported in the literature. For RoTS we use the parameters reported in [Taillard, 1991], for EO we select the parameters reported in [Munera et al., 2016] and for MLS, the only parameter used is the restart process, then no range is needed.

Table 5.2: Parameter ranges of the implemented metaheuristics. (note that n stands for problem instance's size).

Metaheuristic	**Parameter name**	**Range**
RoTS	Tabu duration factor	$[4n$ - $20n]$
	Aspiration factor	$[n^2$ - $10n^2]$
EO	PDF	Power - Exponential - Gamma
	τ	[0,1]
MLS	Start type	Restart from scratch
		Random swaps

5.2.2 GenericTeam Class

As its name suggests, the *GenericTeam* class is the implementation of a generic team. This class encapsulates the main functionality of the PACAS framework. *GenericTeam* contains one *SolutionPopulation* class, one *ParameterControl* class, one *TeamConfiguration* class, and one or many instances of *MultiStartLocalSearch*, *RobustTabuSearch*, *ExtremalOptimization* solvers.

We implemented the master node functionality within the *GeneticTeam* (master description in section 4.2.2). Therefore, the *GeneticTeam* has the functions to establish the number of searcher nodes to run in parallel (`TZ` PACAS parameter), and manage the entry and request policies to the *SolutionPopulation* (`EP` and `RP` PACAS parameters). Finally, *GeneticTeam* carries out one of the most important functions of our framework, the adaptation of the parameters. *GeneticTeam* performances all actions related to the metaheuristic parameters through the *ParameterControl* class. These actions include creating and adapting parameters, evaluating the *performance* based on the *MetaheristicReport* and so on.

J-PACAS supplies three different team types. Their difference lies in the parameter adaptation strategies defined for each one. The *FixedParamsTeam* keeps the parameter constant, i.e. without adaptation during the whole execution of the algorithm. In contrast, the *AdaptedParamsTeam* dynamically changes the parameters every iteration time. These changes are made depending on the performance of each searcher, by comparing the behaviour indicators with the defined diversification limits and solution similarity percentage. To configure the diversification limits, the user must overwrite the method *getDiversifyPercentageLimit* inside the *AdaptedParamsTeam*. This method receives the total number of adaptations and returns an array with the percentages that will be compared in each iteration of the adaptation process. The size

of this array must be equal to the number of adaptations, given that in each adaptation iteration is perform a comparison with the limits. Finally, the *RandomParamsTeam* is a team that performs the adaptation of the parameters randomly. As we will show in the experimental evaluation, a random adaptation of parameters every certain time can be better than keeping them fixed during the whole run.

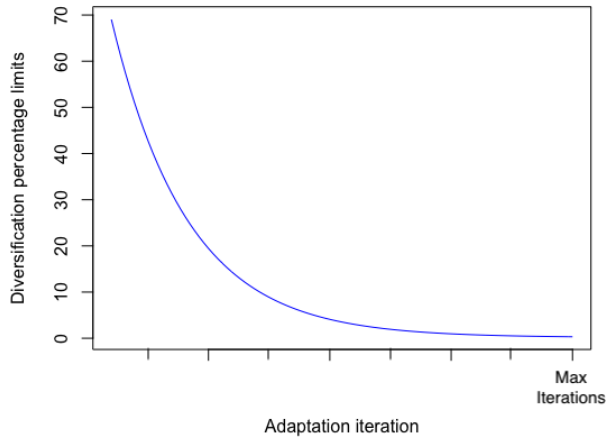

Figure 5.3: J-PACAS Diversification gain percentages limits.

The user's configuration of the team (*AdaptedParamsTeam*) is of vital importance in carrying out a search with a balance between diversification and intensification. After the master node (role fulfilled by the *GeneticTeam*) evaluates the performance of each searcher, it compares this performance against the diversification limits (`DL` PACAS parameter) and the solution similarity percentage (`SSP` PACAS parameter).

The current definition of *getDiversifyPercentageLimit* method samples the exponential function shown in figure 5.3 and which it is defined by the formula 5.1. This sample takes the values needed to satisfy the array size requirement and returns an array of percentages (double values). Through experimentation, we verify that the gain is usually bigger at initial stages of the search than at the final stage. For this reason, we have defined the diversification gain limit during the search process, inspired by how the temperature decreases in simulated annealing [Kirkpatrick et al., 1983]. Using this dynamic limit, J-PACAS diversifies the search more easily at the beginning than at the end of the search process.

$$94.67597e^{-0.31811*iteration} + 0.15699 \tag{5.1}$$

The setting of the `SSP` parameter for the *AdaptedParamsTeam* is located in the class *AlgorithmConfiguration*. In this configuration class there is also the setting for the parameters: iteration time (`I`) and total number of adaptations (`A`) for each team type.

5.2.3 SolutionPopulation Class

An important component of our framework is the Solution Population (SP) and its policies to accept and select solutions. This functionality is completely covered by the class *SolutionPopulation*. The class contains a list of solutions, where there is a solution for each searcher. Its current implementation has 3 types of request policies and 3 types of entry policies. Table 5.3 presents a summary of the request and entry methods and the options available for each one.

Table 5.3: Request and Entry policies in *SolutionPopulation* Class

Method name	**Options**
int[] requestSolution(int)	`RANDOM`: selects a random solution. `SAME`: selects the same solution the searcher has reported. `BEST`: selects the best team solution in the SP.
void entrySolution(Solution, int, QAPData)	`ELITIST`: Entry If the solution cost is better than the worst cost in the SP. `SAME`: Entry solution in the same position that it was selected. `IF_DIFERENT`: Entry if the solution is different to all in SP. If it is not different, it is mutated.

Each team type has its own *SolutionPopulation* class and the configuration of the policies for each team are located in *AlgorithmConfiguration* class.

5.2.4 ParameterControl Class

This class implements methods to manage the metaheuristic parameters. We highlight the main methods. There is one for initializing the parameters of all searcher nodes (*generateInitialParamsPopulation*), another one for evaluating the performance of each metaheuristic by analyzing its report (*getPerformanceEvaluation*) and the last one for adapting the parameters, considering the diversification limits and the solution similarity percentage (*adaptParameter*). Table 5.4 summarizes these methods. These methods are used by *GeneticTeam* to guide each searcher to reach its best parameter settings.

Table 5.4: Main methods in the *ParameterControl* Class

Method name	**Description**
double[] getPerformanceEvaluation(MetaheuristicReport)	Implements the function to evaluate the performance of each searcher node.
int[] adaptParameter(int[] , double[], int, int, int , double[])	Function to adapt the parameters to intensify or diversify.
void generateInitialParamsPopulation(int)	Function to create the initial parameters for each kind of metaheuristic.

The method *getPerformanceEvaluation* receives the *MetaheristicReport* and returns the performance encapsulated in an array. This array contains the indicators

to measure the searcher behavior during the iteration time. These indicators are: the distance between the initial and final solution (**pair-wise difference**) and the **percentage gain** for that iteration.

The method *adaptParameter* receives the performance evaluation, the set of parameters to adapt and other necessary data. This function returns an array of parameters adapted. The adaptation is done according to the indicators, analyzing whether diversification or intensification is needed. For that, the pair-wise difference is computed as similarity comparison criterion. If this distance (as percentage) is lower than `SSP` parameter, J-PACAS considers both solutions as "very similar". Considering it and the percentage gain criteria, within the function *adaptParameter* defines the following rules to determine which action must be taken for adapting the parameters:

- If the gain obtained by the searcher and its pair-wise difference is lower than the corresponding limits, the *adaptParameter* adapts the metaheuristic parameters to *diversify* the search.
- If the gain is higher than the corresponding diversification gain limit or the pair-wise difference is higher than the distance solution limit, the *adaptParameter* adapts the parameters to *intensify* the search.

For each possible case, intensify or diversify, we define actions depending on the metaheuristic type. To perform these actions, *adaptParameter* uses the deltas specified in each metaheuristic and adapts the parameters as follows:

Extremal Optimization: In EO the parameter τ is in the range 0 to 1 and, depending on its value and the PDF, this may lead the metaheuristic to intensify or diversify the search, by adding or subtracting a delta value belonging to the range (see Figure 5.4). The parameters are then adjusted by adding to their values using deltas.

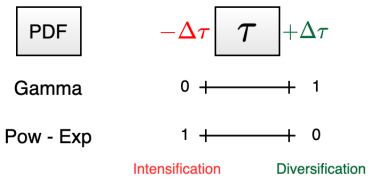

Figure 5.4: Parameters adaptation in EO.

Robust Tabu Search: J-PACAS implemented the parameter adaptation process for RoTS as follows, if the performance analysis indicates *diversify*, a delta of $n/2$ [2] is added to the tabu duration and a delta of $n^2/2$ is added to the aspiration parameters. If the performance analysis indicates *intensify*, the tabu duration is subtracted by $n/2$ and the aspiration is decreased by $n/2$. For intensification, the delta for the adaptation of the aspiration parameter is different from diversification. This is done intending to slow down the intensification process, avoiding to stagnates on a local optimum.

[2] n stands for problem instance's size

Multi-start Local Search: For the case of MLS, if there is any gain in cost, the type of restart is retained. If there is no gain, the *adaptParameter* method changes to the other option.

5.2.5 QAPData Class

The PACAS framework is a conceptual framework used to solve any kind of combinatorial optimization problem. J-PACAS, the PACAS implementation made in Java, provides a class with one of the most difficult optimization problem, the Quadratic Assignment Problem (QAP). This class called *QAPData* inherits from the class *GenericProblem*. The *GenericProblem* class supplies two important methods, the method to make a swap of position for a solution and the method to evaluate a solution. The last one has to be overwritten in the problem specific class, in our case is *QAPData*. *QAPData* also provides other useful methods used in the framework.

Inside the *AlgorithmConfiguration* the user configures the instance of the QAP problem to be solved. The problems set available are the ones in the QAPLIB benchmark, a well-known collection of 134 QAP problems of various sizes and difficulties [Burkard et al., 1991].

5.2.6 AlgorithmConfiguration Class

In this final class, we support the user's configuration for performing a search on a QAP instance. *AlgorithmConfiguration* holds all the variables needed to make this set-up. We divide its structure into two sets of variables. The first set is related to all those general variables of the framework and the second set has to do with those variables related to the configuration of each type of team. The set of general variables includes the problem name, the team size (`TZ`), the timeout in seconds, and other essential data to be configured. The other set is used to define the total number of teams of each type (*FixedParamsTeam*, *RandomParamsTeam*, and *AdaptedParamsTeam*) and their setup. In this setup, there are the specification of the iteration time (`I`), the total number of adaptations (`A`), the request policy (`RP`), the entry policy (`EP`) and the solution similarity percentage (`SSP`).

Table 5.5 depicts the parameters that allow the user configure our implementation of the framework. This table is intrinsically related to table 4.1. Since some parameters in table 4.1 correspond directly to some in our implementation in table 5.5. Table 5.5 classifies variables into **General parameters** and **Team parameters**. There are four types of parameters: integer, string, boolean and integer array. Parameters of type integer array are associated with each type of team. The table shows the integer array names in a short form, these must be completed with the name of the team

[3]Parameters with _____ are incomplete. To complete name write instead of _____ the name of the team to indicate to which type of team the parameter corresponds. The options for the teams are: *FixedParamsTeam*, *RandomParamsTeam* or *AdaptedParamsTeam*.

Table 5.5: Parameters in J-PACAS framework.

Parameter name	Type	Function
	General parameters	
`problemName`	String	Specifies the problem name to solve.
`totalTimeOut`	Integer	Sets the total time in seconds for the whole execution of the algorithm.
`teamSize`	Integer	Establishes the number of searcher nodes per team.
`printResultInConsole`	Boolean	Indicates if the result is showed in console.
`printResultInCSVFile`	Boolean	Indicates if the result is printed in a csv file called 'problemName.csv'.
	Team parameters	
`totalAdaptedParamsTeams`	Integer	Sets the total number of teams *AdaptedParamsTeam*.
`totalRandomParamsTeams`	Integer	Sets the total number of teams *RandomParamsTeam*.
`totalFixedParamsTeams`	Integer	Sets the total number of teams *FixedParamsTeam*.
`iterationTime____` [3]	Integer array	Sets the iteration time for the team indicated in the array position.
`totalAdaptations____`	Integer array	Defines the total parameter adaptations for the team indicated in the array position.
`requestPolicy____`	Integer array	Defines the request policy for the team indicated in the array position.
`entryPolicy____`	Integer array	Defines the entry policy for the team indicated in the array position.
`solutionSimilarityPercertage____`	Integer array	Defines the solution similarity percentage for the team indicated in the array position.

the parameter belongs and user wants to configure. After completing the name, each position of the array indicates the team of the same type to which the parameter refers.

Figure 5.5 displays an example of a configuration of two teams of type *AdaptedParamsTeam*. Thus, the parameter `totalAdaptedParamsTeams` has to be equal to 2. There are then many integer arrays related to this team and must have two values, because each position in the array specifies what team of the two corresponds the value of the parameter. Additionally, in this figure is possible to see how is completed the name of the variables related with the *AdaptedParamsTeam* team. To see more details and examples of the algorithm configuration, please refer to the **README** file [4].

```
// data for team of type: AdaptedParamsTeam
final static int totalAdaptedParamsTeams = 2;
final static int[] iterationTimeAdaptedParamsTeam = { 20000, 15000 };// by iteration in millisecond
final static int[] totalAdaptationsAdaptedParamsTeam = { 15, 20 };
final static int[] requestPolicyAdaptedParamsTeam = { SolutionPopulation.REQUEST_RANDOM,
        SolutionPopulation.REQUEST_SAME };
final static int[] entryPolicyAdaptedParamsTeam = { SolutionPopulation.ENTRY_IF_DIFERENT,
        SolutionPopulation.ENTRY_SAME };
final static int[] solutionSimilarityPercertageAdaptedParamsTeam = { 33, 33 };
```

Figure 5.5: Example parameters configuration of two *AdaptedParamsTeam* teams

5.2.7 Other important classes

We support other important classes in the implementation of our framework. These are the *ManageFile*, *Tools*, *MetaheuristicReport*, *Constructive* and *MainActivity*. The

[4] J-PACAS README file available here: `https://github.com/JonathanDuque/QAPMetaheuristic/blob/framework/README.md`

ManageFile has the methods to read the problem file name data and to write the results obtained in a csv file. The *Tools* class has methods to print relevant information during the algorithm execution. The *Constructive* class has a method to create random solutions. Finally, the *MainActivity* class is the orchestrator of the framework. This class contains all the logic to read and validate the data written in *AlgorithmConfiguration* class. However, its main function is to guide the execution of all the teams, gather their results and finds the best result, i.e. the best solution obtained.

5.3 Conclusion

In this chapter we presented our implementation of the PACAS framework: J-PACAS. This implementation was developed using the *Java* language, which is a suitable language to develop a framework. It complies our requirements, which are: working community, performance, development complexity, standards for parallel programming and programming models supported.

The implementation details of the J-PACAS provide an important guide for the programmers, in order to help them to exploit all features of the implementation or to extend solver methods and functionalities. The users have to configure the parameters showed in table 5.5 to make a complete setup of the framework. In addition, if users decide to modify the diversification limits, they must overwrite the method *getDiversifyPercentageLimit* inside the *AdaptedParamsTeam* class. We plan to extend the ability to create new metaheuritics and teams from the inheritance of generic classes and be able to insert them into the code in a simple way.

In the next chapter, we tackle a real-life combinatorial optimization problem, the QAP. For this, we design several new parallel solvers which are all implemented thanks to J- PACAS.

Chapter 6

Experimental Evaluation

In this section, we present an experimental evaluation of our implementation of the proposed framework. We compare the performance of seven J-PACAS configurations on QAPLIB, a well-known collection of 134 QAP problems of various sizes and difficulties [Burkard et al., 1991]. The instances are named as **nameXX** where **name** corresponds to the first letters of the author and **XX** is the size of the problem. For each instance, QAPLIB also includes the Best Known Solution (BKS), which is sometimes the optimum. Many QAPLIB instances are easy for a parallel solver, we therefore selected the 26 hardest ones for the basic configuration and removed all systematically solved instances in less that 20 seconds, the basic configuration corresponds to an independent parallel method with fixed parameters, called IFIXED.

Each instance is executed 30 times, stopping as soon as the Best Known Solution (BKS) is found or when a time limit of 5 minutes is hit, in case the BKS is not reached. All experiments have been carried out on a quad-AMD Opteron 6380 system, totaling 64 cores running at 2.5GHz and 128 GB of RAM. We ran all J-PACAS configurations under the same conditions and system.

At present, J-PACAS configurations are systematically built with 63 searches nodes: 21 running RoTS, 21 running EO and 21 running MLS and one team. Each searcher node randomly initializes each parameter of its metaheuristic. To do so, a value is randomly picked from the admissible values according to Table 5.2. In all cases, we made sure that each searcher node is actually mapped by the JVM onto a different physical core at runtime.

This section reports an experimentation that was conducted in three phases: the first phase using an independent parallel search strategy, the second a cooperative parallel search and the third using a mixing of strategies for parameter adaptation. There are configurations where the parameters are periodically adapted, parameter control is triggered every 20 seconds. Each metaheuristic can thus adapt its parameters 15 times during the 5 minutes execution cap. Also, there are configurations where the parameters remain fixed during the execution. Our goal is to compare the pre-process parameterization (parameters fixed) with several parameterizations, all together with different request or entry policies to the Solution Population (SP).

We divide six J-PACAS configurations into two groups, the *independent* and the *cooperative*. The difference is the policies used by the groups to manage the SP. For each group, there is a results table. We compare the results of the two groups of J-PACAS configurations, and we create one implementation more, that collects all the best features of the best evaluated configurations. Finally, we compare our best configuration against a PCS state-of-the-art method.

6.1 Independent group evaluation

In the independent group of configurations, the searchers work with the same solution they have ended in each adaptation iteration. This configuration means the searchers perform a parallel independent search without exchanging solutions. To configure this behavior, both request policy and entry policy must have the tag `SAME` for the parameters `requestPolicy____` and `entryPolicy____` corresponding to each team type.

Table 6.1: J-PACAS configurations Independent Group

Parameter name	**IFIXED**	**IRANDOM**	**IADAPTED**
`totalTimeOut (seconds)`	300	300	300
`teamSize`	63	63	63
`totalFixedParamsTeams`	1	0	0
`iterationTimeFixedParamsTeam (milliseconds)`	{20000}	null	null
`totalAdaptationsFixedParamsTeam`	{15}	null	null
`requestPolicyFixedParamsTeam`	{*REQUEST_SAME*}	null	null
`entryPolicyFixedParamsTeam`	{*ENTRY_SAME*}	null	null
`totalRandomParamsTeams`	0	1	0
`iterationTimeRandomParamsTeam (milliseconds)`	null	{20000}	null
`totalAdaptationsRandomParamsTeam`	null	{15}	null
`requestPolicyRandomParamsTeam`	null	{*REQUEST_SAME*}	null
`entryPolicyRandomParamsTeam`	null	{*ENTRY_SAME*}	null
`totalAdaptedParamsTeams`	0	0	1
`iterationTimeAdaptedParamsTeam (milliseconds)`	null	null	{20000}
`totalAdaptationsAdaptedParamsTeam`	null	null	{15}
`requestPolicyAdaptedParamsTeam`	null	null	{*REQUEST_SAME*}
`entryPolicyAdaptedParamsTeam`	null	null	{*ENTRY_SAME*}
`solutionSimilarityPercertageAdaptedParamsTeam`	does not apply	does not apply	{33}

This group contains three J-PACAS configurations, called IFIXED, IRANDOM and IADAPTED. Each configuration is respectively the utilization of each team available (*FixedParamsTeam*, *RandomParamsTeam*, and *AdaptedParamsTeam*). IFIXED configuration keeps the parameter fixed, without adaptation during the whole execution of the algorithm. IRANDOM configuration dynamically changes the parameters randomly every iteration time. IADAPTED configuration changes the parameters dynamically. These changes are made depending on the performance of each searcher. Table 6.1 shows the group configurations and their parameters. The value of each parameter inside the *AlgorithmConfiguration* class is presented. Some values are **null**,

this is because one configuration corresponds to a specific utilization of a team, so the values for the other teams must be null.

Table 6.2 presents the results. For each J-PACAS configuration of this group, the table lists the number of times the BKS is reached across the 30 executions (**#BKS**), the Average Percentage Deviation (**APD**), which is the average of the 30 relative deviation percentages computed as follows: $100 \times \frac{Avg-BKS}{BKS}$, where Avg is the average of the 30 found costs, and finally the average execution time (**Time**). Execution times are given in seconds (as a decimal number). This time is the elapsed (wall clock) time, and includes the time to install all solver metaheuristic instances, solve the problem, communications and the time to detect and propagate the termination.

To compare the performance of all independent configurations, we first compare the number of BKS found, then (in case of tie), the APDs and finally the execution times. For each problem instance, the best-performing configuration row is highlighted and the discriminant field is enhanced in bold font.

Table 6.2: Independent Group Evaluation on 26 hardest instances of QAPLIB.

		IFIXED			IRANDOM			IADAPTED			Wilcoxon Test [1]
	BKS	#BKS	APD	Time	#BKS	APD	Time	#BKS	APD	Time	p-value
sko49	23386	30	0.000	22.7	30	0.000	12.9	30	0.000	**11.5**	-
sko56	34458	30	0.000	34.1	30	0.000	**15.0**	30	0.000	16.4	-
sko64	48498	30	0.000	44.5	30	0.000	**21.8**	30	0.000	25.3	-
sko72	66256	8	0.014	59.6	**27**	0.010	144.2	25	0.010	168.8	0.459
sko81	90998	4	0.016	274.1	**13**	0.010	248.9	10	0.011	257.3	0.358
sko90	115534	1	0.027	299.4	**8**	0.022	277.7	3	0.025	289.6	0.151
sko100a	152002	0	0.037	300.0	6	0.031	269.4	**8**	0.026	256.0	0.146
sko100b	153890	6	0.025	282.5	**15**	0.014	240.6	12	0.012	250.5	0.545
sko100c	147862	6	0.013	300.0	**19**	0.010	282.3	17	0.010	289.1	0.607
sko100d	149576	2	0.026	293.0	4	0.018	294.7	**5**	0.028	284.4	0.099
sko100e	149150	1	0.022	300.0	**7**	0.013	296.6	5	0.015	296.7	0.286
sko100f	149036	1	0.038	299.4	2	0.022	293.2	**3**	0.023	282.0	0.861
tai40a	3139370	1	0.073	293.8	**7**	0.079	277.2	5	0.070	277.3	0.709
tai50a	4938796	2	0.370	286.2	2	**0.352**	289.6	0	0.331	300.0	0.773
tai60a	7205962	0	0.500	300.0	0	**0.440**	300.0	0	0.455	300.0	0.358
tai80a	13499184	0	0.806	300.0	0	0.691	300.0	0	**0.690**	300.0	0.722
tai100a	21044752	0	0.727	300.0	0	0.610	300.0	0	**0.610**	300.0	0.953
tai50b	458821517	30	0.000	29.2	30	0.000	22.3	30	0.000	**18.4**	-
tai60b	608215054	30	0.000	54.2	30	0.000	33.9	30	0.000	**33.8**	-
tai80b	818415043	11	0.013	299.5	**13**	0.045	263.6	5	0.034	290.0	0.292
tai100b	1185996137	0	0.069	300.0	0	0.157	300.0	0	**0.148**	300.0	0.711
tai150b	498896643	0	0.540	300.0	0	0.607	300.0	0	**0.569**	300.0	0.264
lipa90a	360630	30	0.000	27.3	30	0.000	**17.2**	30	0.000	18.1	-
tai256c	44759294	0	0.169	300.0	0	0.166	300.0	0	**0.165**	300.0	0.885
tho150	8133398	0	**0.110**	300.0	0	0.111	300.0	0	0.120	300.0	0.509
wil100	273038	6	0.013	300.0	9	0.019	300.0	**10**	0.021	288.3	0.764
Summary		229	0.139	234.6	**312**	0.132	**219.3**	288	**0.129**	221.2	

[1]Results of the Wilcoxon Signed Rank Test done for comparing the percentage deviation values for the execution of IRANDOM solving a given instance versus the results obtained for IADAPTED.

IRANDOM outperforms IFIXED and IADAPTED on 13 out of 26 of the hardest QAPLIB instances. IADAPTED is the best configuration for 12 instances and IFIXED only for 1 instance. 7 instances remain unsolved (no configuration could obtain any BKS). Clearly, a time limit of 5 minutes is too short for those hard instances. The summary row shows that IRANDOM obtains a better *#BKS* than the others. Respect to IFIXED, IRANDOM increases 36% of BKS (312 vs. 229), and respect to IADAPTED, it increases 8% of BKS (312 vs. 288). However, the average APD got for IRANDOM is not the best of all. The IADAPTED obtains the best APD with 0.129. The other obtains, 0.132 in IRANDOM and 0.139 in IFIXED. It is worth noticing that solutions of better quality are obtained for large instances using IADAPTED and for small instances using IRANDOM.

According to the results obtained, it is more efficient to change the parameters in any way every 20 seconds than to keep them fixed for this benchmark. The benefits of parallelism and adaptation (random or intelligent) guarantee to explored many set of parameter. We use in the experiment 63 searchers, that means 63 parameter sets. Some of these parameters may not be good at a given time, but as the parameters are changing, it is likely that a good set of values is found at some point of the search process, resulting in a better performance.

In summary, IRANDOM obtains more BKS and better results in 13 instances, IADAPTED obtains better APD and better results in 12 instances. Therefore, we consider the performance of these two configurations is similar. To support this conclusion we perform a Wilcoxon Signed Rank Test comparing the percentage deviation values for the execution of IRANDOM solving a given instance versus the results obtained for IADAPTED in the same instance. The null hypothesis or the question we want to address: Is there a difference between the median of cost percentage deviation value of the IADAPTED configuration with respect to IRANDOM in a single instance? Test results are shown in column p-value in Table 6.2.

The p-value presented in Table 6.2 are all greater than 0.05 ($\alpha > 5\%$), thus there is no evidence for rejecting the null hypothesis. We conclude there is no difference between medians of cost percentage deviation values. Hence, the IRANDOM and IADAPTED configurations have a similar performance in the case of the cost percentage deviation of the solutions obtained. Finally, both configurations implement a parallel hybridization strategy and its parameters are adapted. Despite IRANDOM changes its parameters randomly, it selects them from a range taken from state-of-the-art solvers, which report competitive results.

6.2 Cooperative group evaluation

The configurations in the cooperative group manage entry and request policies to the SP focused on cooperation and diversification. To establish cooperation, the `requestPolicy____` parameter of each configuration must have the corresponding tag `RANDOM`. It indicates that each searcher requests a random solution different from

the one reported. To configure diversification through the SP, the `entryPolicy____` parameter of each configuration must have the tag `IF_DIFERENT`. What means that a solution can be entered into the SP if it differs from all others; if it is not different, the solution is mutated to give it a second chance to enter. This behavior allows that inside the SP there always will be different and (possible) good solutions (elite configuration). We compare the performance in the same way as we do for independent configurations.

Cooperative group contains three J-PACAS configurations, CFIXED, CRANDOM and CADAPTED. Again, each configuration is respectively the utilization of each available team (*FixedParamsTeam*, *RandomParamsTeam*, and *AdaptedParamsTeam*). Table 6.3 showed each group's configurations and their parameters. The value of each parameter inside the *AlgorithmConfiguration* class is presented. Some values are **null**, this is because one configuration corresponds to a specific utilization of a team, so the values for the other teams must be null.

Table 6.3: J-PACAS configurations Cooperative Group

Parameter name	**CFIXED**	**CRANDOM**	**CADAPTED**
`totalTimeOut (seconds)`	300	300	300
`teamSize`	63	63	63
`totalFixedParamsTeams`	1	0	0
`iterationTimeFixedParamsTeam (milliseconds)`	{20000}	null	null
`totalAdaptationsFixedParamsTeam`	{15}	null	null
`requestPolicyFixedParamsTeam`	{*REQUEST_RANDOM*}	null	null
`entryPolicyFixedParamsTeam`	{*ENTRY_IF_DIFERENT*}	null	null
`totalRandomParamsTeams`	0	1	0
`iterationTimeRandomParamsTeam (milliseconds)`	null	{20000}	null
`totalAdaptationsRandomParamsTeam`	null	{15}	null
`requestPolicyRandomParamsTeam`	null	{*REQUEST_RANDOM*}	null
`entryPolicyRandomParamsTeam`	null	{*ENTRY_IF_DIFERENT*}	null
`totalAdaptedParamsTeams`	0	0	1
`iterationTimeAdaptedParamsTeam (milliseconds)`	null	null	{20000}
`totalAdaptationsAdaptedParamsTeam`	null	null	{15}
`requestPolicyAdaptedParamsTeam`	null	null	{*REQUEST_RANDOM*}
`entryPolicyAdaptedParamsTeam`	null	null	{*ENTRY_IF_DIFERENT*}
`solutionSimilarityPercertageAdaptedParamsTeam`	does not apply	does not apply	{33}

Table 6.4 presents the results for the cooperative group. CADAPTED outperforms CRANDOM and CFIXED on 12 out of 26 of the hardest QAPLIB instances. CRANDOM is the best configuration for 9 instances and CFIXED only for 5 instances. Regarding the 6 unsolved instances, a time limit of 5 minutes is clearly too short. The summary row shows that CADAPTED obtains a better *#BKS*, 436 vs. 425 in CRANDOM, 436 vs. 432 in CFIXED. CADAPTED also obtains better APD than the others, 0.119 vs. 0.122 in CRANDOM, 0.119 vs. 0.123 in CFIXED. Times are similar.

The overall results are better for the cooperative configurations than the independent ones (Table 6.5). Additionally, the CADAPTED configuration is clearly better when using cooperation. This effect suggests that the adaption of parameters works better when used in conjunction with cooperation. Maybe the parameter adaptation done, depending on the knowledge of the performance of each searcher in one run, is better used in the next run by the new solution exchanged.

According to Table 6.4, the results are mixed. Keeping the parameters fixed or adapting them randomly is better for some instances. CADAPTED and CRANDOM have similar performance. To test if there is any difference between the median of the cost percentage deviation of the CADAPTED configuration with respect to CRANDOM in a single instance, we also perform the Wilcoxon test as done for the independent group. The column p-value in Table 6.4 shows the results of the test. All p-values (greater than 0.05) indicate that the null hypothesis cannot be rejected and there is not difference between medians of cost percentage deviation values. Therefore, CADAPTED and CRANDOM have a similar performance in the cost of the solutions reached.

From the analysis of Table 6.4 results, it can be seen that using different parameter adaptation strategies may perform better. Based on this, we support in PACAS framework a mechanism for mixing different parameter adaptation strategies. We evaluate that through CMIXING configuration, described below (section 6.3).

Table 6.4: Cooperative Group Evaluation on 26 hardest instances of QAPLIB.

		CFIXED			CRANDOM			CADAPTED			Wilcoxon Test [2]
	BKS	#BKS	APD	Time	#BKS	APD	Time	#BKS	APD	Time	p-value
sko49	23386	30	0.000	9.9	30	0.000	11.0	30	0.000	**8.6**	-
sko56	34458	30	0.000	18.7	30	0.000	**16.8**	30	0.000	17.7	-
sko64	48498	30	0.000	21.6	30	0.000	**18.6**	30	0.000	19.4	-
sko72	66256	30	0.000	101.9	30	0.000	**92.2**	30	0.000	97.7	-
sko81	90998	25	0.010	**154.3**	23	0.010	184.7	25	0.010	188.5	0.529
sko90	115534	14	0.011	229.9	13	0.012	268.6	**16**	0.013	215.1	0.454
sko100a	152002	**20**	0.017	204.3	11	0.022	248.2	11	0.021	254.6	0.945
sko100b	153890	25	0.010	193.2	**28**	0.010	197.9	23	0.010	201.6	0.075
sko100c	147862	27	0.010	277.8	**29**	0.010	268.8	27	0.010	279.3	0.313
sko100d	149576	18	0.014	253.5	15	0.012	279.9	**19**	0.011	251.1	0.282
sko100e	149150	26	0.010	205.5	26	0.010	213.3	**28**	0.010	238.6	0.401
sko100f	149036	8	0.016	265.1	11	0.015	256.3	11	**0.013**	240.6	0.730
tai40a	3139370	7	0.074	266.0	6	0.076	262.5	7	**0.070**	267.5	0.348
tai50a	4938796	0	0.377	300.0	1	**0.329**	298.7	1	0.356	293.3	0.976
tai60a	7205962	0	**0.445**	300.0	0	0.449	300.0	0	0.458	300.0	0.756
tai80a	13499184	0	0.696	300.0	0	0.715	300.0	0	**0.696**	300.0	0.314
tai100a	21044752	0	**0.618**	300.0	0	0.624	300.0	0	0.621	300.0	0.745
tai50b	458821517	30	0.000	19.7	30	0.000	**19.2**	30	0.000	22.4	-
tai60b	608215054	30	0.000	29.8	30	0.000	28.6	30	0.000	**27.1**	-
tai80b	818415043	17	0.025	270.7	17	0.023	243.7	**19**	0.025	255.5	0.731
tai100b	1185996137	11	0.044	275.3	12	0.048	271.9	**16**	0.045	253.4	0.345
tai150b	498896643	0	0.546	300.0	0	0.561	300.0	0	**0.491**	300.0	0.076
lipa90a	360630	30	0.000	27.0	30	0.000	20.5	30	0.000	**20.4**	-
tai256c	44759294	0	0.172	300.0	0	**0.166**	300.0	0	0.170	300.0	0.503
tho150	8133398	0	0.081	300.0	0	**0.070**	300.0	0	0.076	300.0	0.474
wil100	273038	**24**	0.010	278.4	23	0.010	289.0	23	0.010	288.6	1.000
Summary		432	0.123	**200.1**	425	0.122	203.5	**436**	**0.119**	201.6	

[2]Results of the Wilcoxon Signed Rank Test done for comparing the percentage deviation values for the execution of CADAPTED solving a given instance versus the results obtained for CRANDOM.

Notice that cooperative group configurations are better than independent configurations. If we compare each cooperative with its independent equivalent (e.g. CRANDOM vs. IRANDOM), we can see that cooperative configuration achieves better results, at least a 36% increase in the number of optima, at least a 7.58% less APD and at least a 7.2% less time (see Table 6.5). The effect of cooperation behavior has direct impact on the performance of each configuration, with or without any parameterization.

Table 6.5: Comparison performance: Independent vs Cooperative

Configuration	**#BKS**	**APD**	**Time**
IFIXED	229	0.139	234.6
CFIXED	432	0.123	200.1
Improvement	88%	11.51%	14.7%
IRANDOM	312	0.132	219.3
CRANDOM	425	0.122	203.5
Improvement	36%	7.58%	7.2%
IADAPTED	288	0.129	221.2
CADAPTED	436	0.119	201.6
Improvement	51%	7.75%	8.9%
Summary Independent	829	0.400	675.1
Summary Cooperative	1293	0.364	606.2
Improvement	56%	9%	10.2%

6.3 Evaluation of mixing parameter adaptation strategies

We also evaluated the mixing of different parameter adaptation strategies. According to the performance of the cooperative group (CFIXED, CRANDOM and CADAPTED) we proposed inside our PACAS framework the implementation of this combination of strategies. This configuration, CMIXING maintains the same request and entry policy values as the cooperative group to manage the SP. CMIXING configuration is characterized for having one third of the searchers using parameters fixed, other third using adapted parameters and the remaining third with random parameters. To select which adaptation strategy the searcher will use, after the first iteration time, J-PACAS organizes the parameters from best to worst, depending on the gain in cost of the objective function reached for a specific parameter. Then, the best third keeps fixed, the second best third are adapted, and the worst third are randomly changed.

Table 6.6 presents the results for CMIXING. We compared the results with CFIXED, since we used it as our control configuration. To evaluate whether there are differences

between the percentage deviation of the cost of the configurations, we perform again the Wilcoxon Test. Column p-value in Table 6.6 shows the test result.

According to the p-values, there are differences performances in the percentage deviation obtained in 4 instances (p-value less than 0.05). For three instances CMIXING presents a better performance than CFIXED with statistical significance, only in one instance is the contrary. In the other 24 instances there is no difference between medians of cost percentage deviation values, however we can analyse the complementary results presented in the table (BKS, APD and times). CMIXING outperforms CFIXED in most instances, CMIXING reaches more BKS (437 vs. 432) and, when the optimum is not reached, solutions provided by CMIXING are of much better quality than CFIXED as shown by the APDs (0.116 vs. 0.123), and it does so in a shorter period of time. Clearly, the use of different adaptation strategies can result in better performance.

Observe that CFIXED is indeed an efficient solver for this benchmark, it implements a parallel cooperative hybridization strategy and its parameters, despite being fixed, are picked within a range taken from state-of-the-art solvers which report competitive results.

6.4 Distribution analysis of winning metaheuristics

It is interesting to analyze the effect of the characteristics on the embedded metaheuristics forming each hybrid solver configuration. The intended behaviour is that, in the cooperative group, all metaheuristics contribute to solve the problem. Different from the independent group, where the stronger metaheuristic for the QAPLIB instances is more likely to find the best solution, even if all should have their chance. Also, it is important to verify that our PACAS parameter adaptation does not privilege one metaheuristic over the others.

Table 6.7 presents the distribution of "winning" metaheuristics (as percentages) for all configurations cooperative and independent on the QAPLIB benchmark. We differentiate the cooperative and independent configurations as FIXED, RANDOM and ADAPTED. The winner metaheuristic is RoTS which finds the best solution in all cases of any configuration, 80.13% in independent and 44.57% with cooperation; followed by EO and by MLS. This can be explained by the effectiveness of RoTS which performs very well on QAP (it is particularly efficient in the intensification phases). The difference between RoTS and the other metaheuristics is bigger in the independent than in the cooperative group. However, in the cooperative group, the winner is more distributed, EO and MLS are also interesting candidates thanks to the cooperation and their intrinsic ability to escape local minima and thus to diversify the search.

Table 6.8 introduces the distribution of "winning" metaheuristics and "winning" parameter configuration (all as percentages) for CMIXING configuration. Regarding to parameter configuration, when the best is achieved in less than 20 seconds we call the parameter configuration **Initial** (no adaption is performed), when it is reached with

Table 6.6: Mixing parameter adaptation strategies, Evaluation on 26 hardest instances of QAPLIB.

		CFIXED			CMIXING			Wilcoxon Test
	BKS	#BKS	APD	Time	#BKS	APD	Time	p-value
sko49	23386	30	0.000	9.9	30	0.000	**9.1**	-
sko56	34458	30	0.000	18.7	30	0.000	**13.8**	-
sko64	48498	30	0.000	21.6	30	0.000	**20.6**	-
sko72	66256	30	0.000	101.9	30	0.000	**86.7**	-
sko81	90998	**25**	0.010	154.3	24	0.010	158.2	0.749
sko90	115534	14	0.011	229.9	**15**	0.012	221.9	0.821
sko100a	152002	**20**	0.017	204.3	8	0.024	268.5	**0.000**
sko100b	153890	25	0.010	193.2	**28**	0.010	187.2	0.237
sko100c	147862	27	0.010	277.8	27	0.010	**257.5**	1.000
sko100d	149576	**18**	0.014	253.5	16	0.014	273.7	0.665
sko100e	149150	26	0.010	205.5	26	0.010	**205.0**	1.000
sko100f	149036	8	0.016	265.1	**16**	0.015	230.3	**0.043**
tai40a	3139370	**7**	0.074	266.0	3	0.072	278.7	0.365
tai50a	4938796	0	0.377	300.0	**1**	0.332	294.4	**0.043**
tai60a	7205962	0	0.445	300.0	0	**0.434**	300.0	0.767
tai80a	13499184	0	0.696	300.0	0	**0.641**	300.0	**0.031**
tai100a	21044752	0	0.618	300.0	0	**0.607**	300.0	0.441
tai50b	458821517	30	0.000	**19.7**	30	0.000	21.4	-
tai60b	608215054	30	0.000	29.8	30	0.000	**26.4**	-
tai80b	818415043	17	0.025	270.7	**20**	0.023	234.2	0.392
tai100b	1185996137	11	0.044	275.3	**17**	0.050	264.6	0.288
tai150b	498896643	0	0.546	300.0	0	**0.512**	300.0	0.287
lipa90a	360630	30	0.000	27.0	30	0.000	**18.2**	-
tai256c	44759294	0	0.172	300.0	0	**0.169**	300.0	0.824
tho150	8133398	0	0.081	300.0	0	**0.067**	300.0	0.073
wil100	273038	24	0.010	278.4	**26**	0.010	273.1	0.499
Summary		432	0.123	200.1	**437**	**0.116**	**197.8**	

Table 6.7: Distribution of winning metaheuristics in cooperative and independent group (analysis).

	Independent			**Cooperative**		
	RoTS	EO	MLS	RoTS	EO	MLS
FIXED	84.62%	10.77%	4.62%	44.23%	37.82%	17.95%
RANDOM	77.82%	10.38%	11.79%	43.85%	36.41%	19.74%
ADAPTED	77.95%	10.64%	11.41%	45.64%	36.03%	18.33%
Summary	80.13%	10.60%	9.27%	44.57%	36.75%	18.67%

fixed parameters we call it **Best**, and when it is achieved with adapted parameters, we call it **Adapted** or **Random**. In CMIXING, the winner metaheuristic behaves like the cooperative group in Table 6.7.

Table 6.8: Distribution of winning on CMIXING configuration

Winner Metaheuristic			**Winner parameter strategy**			
RoTS	EO	MLS	Initial	Best	Adapted	Random
45.26%	37.18%	17.56%	12.44%	36.15%	28.85%	22.56%

Respect to the winner parameter strategy, the winner is "Best", it obtains 36.15% of the success cases, followed by "Adapted" with 28.85% and "Random" with 22.56%. The worst is "Initial" with 12.44%, as expected, since it is hard to find the best solution in less than 20 seconds. It is s a good strategy identifying good parameters and keeping these fixed, meanwhile the others are adapted. For instance, F-Race, a Parameter Tuning Strategy that uses a race algorithm, behaves similarity to CMIXING. In F-Race, some parameters are discarded based on a statistical study and those who pass the test continue the race. In our CMIXING implementation, those parameters identified as "good" continue during the algorithm execution, the others are intelligently adapted or randomly modified.

In summary, in independent or cooperative group configuration, the winner keeps its winning percentage belongs to the group. In cooperative, RoTS around 44%, EO around 36% and MLS around 18%, CMIXING is included here. In independent group, RoTS around 80%, EO around 10% and MLS around 9%. We conclude, the J-PACAS parameter adaptation process does not affect the well behavior of each metaheuristics, in any of the seven configurations. RoTS who performs very well on QAP proves its efficiency being in most cases the winner and when there is cooperation, EO and MLS

contribute to find better results.

6.5 Comparison with a state-of-the-art method

We compare our best result configuration CMIXING with one similar state-of-the-art PCS method that performs online parameterization in a parallel hybrid method; it is the Multistart Hyper-heuristic Algorithm for the QAP (MSH-QAP) [Dokeroglu and Cosar, 2016]. We compare the performance of CMIXING and MSH-QAP in the 26 QAP instances we selected, we develop the comparison following the same process as above.

MSH-QAP uses, as low-level heuristics, some of the metaheuristics that have been reported to be among the best performance for large problem instances of the QAP, Simulated Annealing, Robust Tabu Search, Fast Ant System, and Breakout Local Search. MSH-QAP has two execution phases. In the first phase, a genetic algorithm (GA) performs the role of high-level heuristic. In this phase, the GA selects the best metaheuristics and tunes their parameters. The metaheuristic methods are executed in parallel. The parameters used for the metaheuristic algorithms are adjusted adaptively by using crossover and mutation operators. They also use operators for selecting the metaheuristics. If they do not find an optimum solution in the first phase, the second phase is activated. Then, in that phase, the best metaheuristic (with its parameter setting) is run on several processors with multiple starts.

The MSH-QAP's experiments are performed on a High Performance Cluster computer which has 46 nodes, each with 2 CPUs giving 92 CPUs. Each CPU has 4 cores, giving a total of 368 cores. Each node has 16 GB of RAM giving 736 GB of total memory. MSH-QAP was developed using *C++* and the MPI libraries. These experiments were done using 64 cores in total and executing each problem instance 10 times. The authors reported the same data we reported, but times in minutes (as a decimal number). The timeout is not clear; it seems to depend on the problem size, given that for large instance the timeout is 75 minutes, but for others is less. It also is not clear at what moment the phase two is activated. Additionally, the results for `tai150b`, `tai256c`, `tho150` and `wil100` are not reported in the paper.

Making a fair comparison is difficult, because the execution conditions are different, this is a big issue in the metaheuristic community. However, some conclusions can be drawn. First, we make a comparison of those problems that were solved in a timeout of 5 minutes by MSH-QAP, then we compare the remaining ones.

Table 6.9 presents the results of CMIXING vs. MSH-QAP for 11 problem instances solved in a timeout of 5 minutes. CMIXING outperforms MSH-QAP on 7 out of the 11 instances. MSH-QAP also solved these 7 instances, but it took more time than CMIXING. MSH-QAP is the best method in 4 instances. In these 4, CMIXING is indeed a good solver, since it reached at least 15 BKS in a less time.

In summary, CMIXING had 86.7% of success with respect to the total BKS that can be obtained and MSH-QAP 100% of success. CMIXING despite having a lower

Table 6.9: Comparison results CMIXING vs. MSH-QAP (Timeout 5 mins).

		CMIXING			MSH-QAP		
	BKS	#BKS	APD	Time	#BKS	APD	Time
sko49	23386	30	0.000	**0.15**	10	0.000	2.8
sko56	34458	30	0.000	**0.23**	10	0.000	2.8
sko64	48498	30	0.000	**0.34**	10	0.000	3.2
sko72	66256	30	0.000	**1.44**	10	0.000	3.6
sko81	90998	24	0.010	2.64	**10**	0.000	4.1
sko90	115534	15	0.012	3.7	**10**	0.000	4.5
tai50b	458821517	30	0.000	**0.36**	10	0.000	3
tai60b	608215054	30	0.000	**0.44**	10	0.000	3.2
tai80b	818415043	20	0.023	3.9	**10**	0.000	4
tai100b	1185996137	17	0.050	4.41	**10**	0.000	5
lipa90a	360630	30	0.000	**0.3**	10	0.000	4.5
Summary		286 (86.7%)	0.009	**1.63**	110 (**100%**)	**0.000**	3.7

percentage of BKS is faster than MSH-QAP (97.7 vs. 222) and it has a small APD value (0.009).

It is worth noting that, for CMIXING we do not present 108 QAP instances that were solved in less than 20 seconds with a 100% of success. MSH-QAP solves these instances with 100% of success too, but taking at least 36 seconds (0.6 minutes).

Table 6.10 shows the results of CMIXING vs. MSH-QAP for the other 11 problem instances. MSH-QAP solved these 11 instances with a time limit much greater than 5 minutes. We kept its times in minutes, to show more clearly the big differences in execution times with CMIXING. CMIXING outperforms MSH-QAP on 6 out of the 11 instances, using a time limit of only 5 minutes. MSH-QAP is the best method for the remaining 5 instances, but using a greater time limit of 45-75 minutes.

Our experiments for the `tai50a` and `tai100a` instances are compared against an updated (smaller) BKS cost. In these cases, CMIXING was better for `tai50a` and MSH-QAP was better for `tai100a`, but the latter performed an execution on a very high time (75 min). Meanwhile, CMIXING only used 5 minutes. Regarding to the 3 unsolved instances, MSH-QAP was better, however CMIXING got as a maximum APD of 0.641 in the total execution time.

In summary, CMIXING had 37.9% of success in finding the BKS and MSH-QAP 18.2% of success. The BKS obtained by CMIXING are in much less time (263.2 sec or 4.4 min vs. 63.4 min), which indicates that CMIXING is an adequate J-PACAS configuration to obtain good results in short times for this benchmark. It is very competitive method, and it is comparable with one of the best state-of-the art method, as MSH-QAP.

Table 6.10: Comparison results CMIXING vs. MSH-QAP (Timeout greater than 5 min).

		CMIXING			MSH-QAP		
	BKS	#BKS	APD	Min	#BKS	APD	Min
sko100a	152002	**8**	0.024	4.48	0	0.003	75
sko100b	153890	**28**	0.010	3.12	0	0.004	75
sko100c	147862	**27**	0.010	4.29	0	0.003	75
sko100d	149576	**16**	0.014	4.56	0	0.004	75
sko100e	149150	26	0.010	3.42	**10**	0.000	75
sko100f	149036	16	0.015	3.84	**10**	0.000	75
tai40a	3139370	**3**	0.072	4.65	0	0.261	30
tai50a	4941410 (4938796)	**1**	0.332	4.91	0	0.165	37.5
tai60a	7205962	0	0.434	5	0	**0.270**	45
tai80a	13499184	0	0.641	5	0	**0.530**	60
tai100a	21059006 (21044752)	0	0.607	5	0	**0.338**	75
Summary		125 (**37.9%**)	0.197	4.39	20 (18.2%)	**0.143**	63.4

6.6 Conclusion

We have reported the results of seven J-PACAS configurations evaluated on the 26 hardest instances of QAPLIB. In view of the results, CMIXING is the recommended configuration for getting good results for this benchmark using as a maximum execution time 5 minutes.

We have compared the CMIXING results (our best) against MSH-QAP. MSH-QAP is a parallel hybrid solver that uses four different metaheuristic methods. The adaptation of the parameters in MSH-QAP is carried out by a GA, the GA dynamically changes the parameters through crossover and mutation operators. The comparison results showed that CMIXING outperforms MSH-QAP in most of the cases taking a shorter time than MSH-QAP. The adaptation of the parameter in CMIXING is most suitable that the adaptation done in MSH-QAP, for solving the QAPLIB instances.

Chapter 7

Conclusion

To configure the parameters of metaheuristic methods we can resort to parameter tuning (PTS), parameter control (PCS) and instance-specific parameter tuning strategies (IPTS). The creation of parallel hybrid metaheuristic methods requires also the fine setting of a larger number of parameters, since more metaheuristics (of different types) are involved and parallel behavior involves yet another set of parameters. Tuning this methods increase number of parameters makes it even more difficult to find the appropriate setting for the algorithm to behave optimally. To automate this task, we have proposed PACAS, a framework to configure the **PA**rameter **C**ontrol **A**daptation for **S**ingle solution metaheuristics in a parallel hybrid solver for the efficient solution of combinatorial optimization problems (COP).

The PACAS framework aimed at increasing the performance of parallel method based on several parameter control strategies through the analysis of the behavior of single solution metaheuristic. Meanwhile hybrid behavior is granted by the use of different types of metaheuristics and by the cooperation through the intra-team communication. This last feature is called hybridization through cooperative parallelism.

We proposed an implementation of the PACAS framework using the Java language: J-PACAS. J-PACAS provides three different metaheuristic: Robust Tabu Search (RoTS), Extremal Optimization (EO), and Multistart Local Search (MLS). To hybridize the cooperative parallel behavior different request and entry policies to manage the solution population are supported. J-PACAS supplies three different kind of teams, their difference lies in the parameter adaptation strategies defined for each one. These strategies are: keep the parameter fixed, adapt the parameters intelligently or adapt them in a random way. We used this implementation to validate our framework, tackling one of the most difficult COPs, the QAP.

Our implementation allows to adapt the metaheuristic parameters, numerical or categorical through deltas. This adaptation is carry out according to a performance evaluation, where an analysis of the behavior of the single solution metaheuristic is done.

7.1 Research perspectives

In the PACAS framework, only one communication scheme has been implemented within each team; as future work, an extension to the framework can be made to enable communication between teams to take greater advantage of cooperative parallelism.

Up to now, J-PACAS has incorporated a RoTS, an EO and a MLS. In addition, only the class for the QAP has been developed. Another line of potential improvements in the medium-term work is the extension of J-PACAS by embedding new metaheuristics, maybe a population-based metaheuristics, such as genetic algorithm. We plan to implement other models of COPs.

J-PACAS allows various configurations through a set of parameters. Future work in the medium-term is to analyze the effect on J-PACAS performance of varying the parameters related to the parameter control strategy, possibilities abound. For instance, to change the total adaptations and the iteration time (parameters `A` and `I`), testing many deltas to intensify or diversify the search (`DF` and `DI`), to analysis several iterations without improvement (`IWI`) and to experiment with different diversification limits and solution similarity percentage (`DL` and `SSP`). We also suggest, as a future work to study the effect done by a PCS configuration on a specific metaheuristic.

The experimental evaluation carried out in this research indicate that together, the parameter adaptation and the cooperative parallelism provided by J-PACAS performs well in the most difficult instances of QAPLIB. Future work in the short term is to test J-PACAS in other difficult QAP instances, such as those proposed by Drezner [2005] and Palubeckis [2000], which were designed to be especially difficult to solve using metaheuristics. Finally, given that there are difficult instances of QAPLIB where a time limit of 5 minutes is too short. We will experiment on machines with more cores, with other architectures and with time limits greater than 5 minutes.

Acknowledgments

Today this adventure is close to finishing and I need to thank everyone that helped me along the way.

I would like to thank to my thesis director Danny Múnera for the support and guidance he has provided, for helping me to organize my ideas, for spending time and for his excellent advices regarding to all the academic process in my master. We had a difficult time living through the COVID pandemic, but he was always patient with the delays in my process.

I would like to thank to the Professors Salvador Abreu and Daniel Diaz for their help in presenting the results of my research to the academic community and for allowing me to work in the computer clusters of their workplaces, the University of Evora in the case of Professor Salvador and Paris 1 in the case of Professor Daniel.

I would like to thank to GITA (Grupo de Investigación en Telecomunicaciones Aplicadas) for giving me a work space in the laboratory to carry out my studies.

Last but not least, I want to thank my mother Dora, she always support me, even if she had no idea what I was doing.

This work is dedicated to the memory of my best friend Danilo Herrera. He could not see me finish this step in my life. His words were always important for me to keep going and still are.

I would like to thank my girlfriend, Camila Rodriguez, who always supported me during the hardest times. Thank your understanding when I did not have time. I started this adventure when we met, now we are a couple.

I would like to thank to my GITA colleagues and master classmates, thanks for the coffees and discussions.

Finally, this research was supported by the CODI (*Comité para el desarrollo de la Investigación*) project PRV19-1-01 funded by the University of Antioquia in Medellín, Colombia.

Bibliography

B. Adenso-Diaz and M. Laguna. Fine-tuning of algorithms using fractional experimental designs and local search. *Operations research*, 54(1):99–114, 2006.

E. Alba, G. Luque, and S. Nesmachnow. Parallel metaheuristics: Recent advances and new trends. *International Transactions in Operational Research*, 20(1):1–48, 2013. ISSN 09696016. doi: 10.1111/j.1475-3995.2012.00862.x.

C. Ansótegui, M. Sellmann, and K. Tierney. A gender-based genetic algorithm for the automatic configuration of algorithms. In *International Conference on Principles and Practice of Constraint Programming*, pages 142–157. Springer, 2009.

P. Balaprakash, M. Birattari, and T. Stützle. Improvement strategies for the f-race algorithm: Sampling design and iterative refinement. In *International workshop on hybrid metaheuristics*, pages 108–122. Springer, 2007.

E. B. Barbosa and E. L. F. Senne. A heuristic for optimization of metaheuristics by means of statistical methods. In *ICORES*, pages 203–210, 2017.

T. Bartz-Beielstein, C. W. Lasarczyk, and M. Preuß. Sequential parameter optimization. In *2005 IEEE congress on evolutionary computation*, volume 1, pages 773–780. IEEE, 2005.

R. Battiti and M. Brunato. Reactive search optimization: learning while optimizing. In *Handbook of Metaheuristics*, pages 543–571. Springer, 2010.

R. Battiti and G. Tecchiolli. The reactive tabu search. *ORSA journal on computing*, 6(2):126–140, 1994.

D. Beasley, D. R. Bull, and R. R. Martin. An overview of genetic algorithms: Part 1, fundamentals, 1993. URL `http://orca.cf.ac.uk/64436/1/ga{_}overview1.pdf`.

R. K. Bhati and A. Rasool. Quadratic Assignment Problem and its Relevance to the Real World: A Survey. *International Journal of Computer Applications*, 96(9): 42–47, 2014.

M. Birattari and J. Kacprzyk. *Tuning metaheuristics: a machine learning perspective*, volume 197. Springer, 2009.

M. Birattari, T. Stützle, L. Paquete, K. Varrentrapp, et al. A racing algorithm for configuring metaheuristics. In *Gecco*, volume 2, 2002.

M. Blesa and F. Xhafa. Using parallelism in experimenting and fine tuning of parameters for metaheuristics. In *International Conference on Computational Science*, pages 429–432. Springer, 2004.

C. Blum, A. Roli, and M. Sampels. *Hybrid metaheuristics: an emerging approach to optimization*, volume 114. Springer, 2008.

C. Blum, J. Puchinger, G. R. Raidl, and A. Roli. Hybrid metaheuristics in combinatorial optimization: A Survey. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 7505 LNCS(6):1–10, 2011. ISSN 03029743. doi: 10.1007/978-3-642-33860-1_1. URL http://dx.doi.org/10.1016/j.asoc.2011.02.032.

S. Boettcher and A. Percus. Nature's way of optimizing. *Artificial Intelligence*, 119(1):275–286, 2000. ISSN 0004-3702. doi: https://doi.org/10.1016/S0004-3702(00)00007-2. URL http://www.sciencedirect.com/science/article/pii/S0004370200000072.

I. Boussaïd, J. Lepagnot, and P. Siarry. A survey on optimization metaheuristics. *Information Sciences*, 237:82–117, 2013. ISSN 00200255. doi: 10.1016/j.ins.2013.02.041.

R. E. Burkard. Quadratic assignment problems. *European Journal of Operational Research*, 15(3):283–289, 1984. ISSN 0377-2217. doi: https://doi.org/10.1016/0377-2217(84)90093-6. URL http://www.sciencedirect.com/science/article/pii/0377221784900936.

R. E. Burkard, S. Karisch, and F. Rendl. QAPLIB - a Quadratic Assignment Problem Library. *European Journal of Operational Research*, 55(1):115–119, 1991. ISSN 03772217. doi: 10.1016/0377-2217(91)90197-4. URL http://www.sciencedirect.com/science/article/pii/0377221791901974.

R. E. Burkard, M. Dell'Amico, and S. Martello. *Assignment Problems, Revised Reprint*. Other Titles in Applied Mathematics. Society for Industrial and Applied Mathematics, 2012. ISBN 161197223X. URL https://books.google.com/books?hl=en{&}lr={&}id=dyQR2ElvTTUC{&}pgis=1.

E. K. Burke, G. Kendall, J. Newall, E. Hart, P. Ross, and S. Schulenburg. Hyper-Heuristics: An Emerging Direction in Modern Search Technology BT. In F. Glover and G. A. Kochenberger, editors, *Handbook of Metaheuristics*, pages 457–474. Springer US, Boston, MA, 2003. ISBN 978-0-306-48056-0. doi: 10.1007/0-306-48056-5_16. URL https://doi.org/10.1007/0-306-48056-5{_}16.

E. K. Burke, M. R. Hyde, G. Kendall, G. Ochoa, E. Ozcan, and J. R. Woodward. Exploring Hyper-heuristic Methodologies with Genetic Programming. pages 177–201, 2009. doi: 10.1007/978-3-642-01799-5_6.

E. K. Burke, M. Gendreau, M. R. Hyde, G. Kendall, G. Ochoa, E. Özcan, and R. Qu. Hyper-heuristics: A survey of the state of the art. *Journal of the Operational Research Society*, 64(12):1695–1724, 2013. ISSN 01605682. doi: 10.1057/jors.2013.71.

E. K. Burke, M. R. Hyde, G. Kendall, G. Ochoa, E. Özcan, and J. R. Woodward. A Classification of Hyper-Heuristic Approaches : Revisited. 2019.

L. Calvet, A. A. Juan, C. Serrat, and J. Ries. A statistical learning based approach for parameter fine-tuning of metaheuristics. *SORT-Statistics and Operations Research Transactions*, pages 201–224, 2016.

Y. Caniou, P. Codognet, F. Richoux, D. Diaz, and S. Abreu. Large-scale parallelism for constraint-based local search: the costas array case study. *Constraints*, 20(1):1–27, 2014. ISSN 1383-7133, 1572-9354. doi: 10.1007/s10601-014-9168-4. URL `http://link.springer.com/article/10.1007/s10601-014-9168-4$\backslash$nhttp://link.springer.com/article/10.1007/s10601-014-9168-4{#}close`.

K. Chakhlevitch and P. Cowling. Hyperheuristics: Recent developments. *Handbook of Heuristics*, (June 2008):1–19, 2008. ISSN 1860-949X. doi: 10.1007/978-3-540-79438-7.

P. Codognet, D. Munera, D. Diaz, and S. Abreu. Parallel local search, 2018.

C. W. Commander. A survey of the quadratic assignment problem, with applications. *Morehead Electronic Journal of Applicable Mathematics*, 4:MATH–2005–01, 2005.

P. Cowling, G. Kendall, and E. Soubeiga. A Hyperheuristic Approach to Scheduling a Sales Summit BT - Practice and Theory of Automated Timetabling III. pages 176–190, Berlin, Heidelberg, 2001. Springer Berlin Heidelberg. ISBN 978-3-540-44629-3.

S. P. Coy, B. L. Golden, G. C. Runger, and E. A. Wasil. Using experimental design to find effective parameter settings for heuristics. *Journal of Heuristics*, 7(1):77–97, 2001.

T. Crainic and M. Toulouse. Parallel Meta-Heuristics. In M. Gendreau and J.-Y. Potvin, editors, *Handbook of Metaheuristics*, volume 146 of *International Series in Operations Research & Management Science*, pages 497–541. Springer US, 2010.

C. Dai, Y. Zhu, and W. Chen. Adaptive probabilities of crossover and mutation in genetic algorithms based on cloud model. In *2006 IEEE Information Theory Workshop-ITW'06 Chengdu*, pages 710–713. IEEE, 2006.

K. A. De Jong. Analysis of the behavior of a class of genetic adaptive systems. Technical report, 1975.

J. Denzinger, M. Fuchs, and M. Fuchs. *High Performance ATP Systems by Combining Several AI Methods*, volume 1. dec 1996.

F. Dobslaw. A parameter tuning framework for metaheuristics based on design of experiments and artificial neural networks. In *International Conference on Computer Mathematics and Natural Computing*. WASET, 2010a.

F. Dobslaw. Recent development in automatic parameter tuning for metaheuristics. In *Proceedings of the 19th Annual Conference of Doctoral Students-WDS 2010*, 2010b.

T. Dokeroglu and A. Cosar. A novel multistart hyper-heuristic algorithm on the grid for the quadratic assignment problem. *Engineering Applications of Artificial Intelligence*, 52:10–25, 2016. ISSN 09521976. doi: 10.1016/j.engappai.2016.02.004.

K. Dowsland and A. Diaz. Heuristic design and fundamentals of the Simulated Annealing. *Inteligencia Artificial*, 7(19):93–102, 2003. ISSN 1137-3601. doi: 10.4114/ia.v7i19.718.

Z. Drezner. The Extended Concentric Tabu for the Quadratic Assignment Problem. *European Journal of Operational Research*, 160(2):416–422, 2005. ISSN 03772217.

J. Duque, D. A. Múnera, D. Díaz, and S. Abreu. Solving qap with auto-parameterization in parallel hybrid metaheuristics. In *International Conference on Optimization and Learning*, pages 294–309. Springer, 2021.

Á. E. Eiben, R. Hinterding, and Z. Michalewicz. Parameter control in evolutionary algorithms. *IEEE Transactions on evolutionary computation*, 3(2):124–141, 1999.

A. E. Eiben, Z. Michalewicz, M. Schoenauer, and J. E. Smith. Parameter control in evolutionary algorithms. In *Parameter setting in evolutionary algorithms*, pages 19–46. Springer, 2007.

E. Ferro, J. Santos, J. Orozce, and R. Cayssials. Tuning parameters to improve a heuristic method: More and better solutions to an np-hard real-time problem. In *Proceedings of the Eighth Euromicro Workshop on Real-Time Systems*, pages 47–51. IEEE, 1996a.

E. Ferro, J. Santos, J. Orozce, and R. Cayssials. Tuning parameters to improve a heuristic method: More and better solutions to an np-hard real-time problem. In *Proceedings of the Eighth Euromicro Workshop on Real-Time Systems*, pages 47–51. IEEE, 1996b.

Á. Fialho. *Adaptive operator selection for optimization*. PhD thesis, Université Paris Sud-Paris XI, 2010.

R. A. Fisher. Design of experiments. *Br Med J*, 1(3923):554–554, 1936.

T. C. Fogarty. Varying the probability of mutation in the genetic algorithm. In *Proceedings of the 3rd International Conference on Genetic Algorithms*, pages 104–109, 1989.

L. J. Fogel, A. J. Owens, and M. J. Walsh. Artificial intelligence through simulated evolution. 1966.

P. Garrido and M. C. Riff. DVRP: A hard dynamic combinatorial optimisation problem tackled by an evolutionary hyper-heuristic. *Journal of Heuristics*, 16(6):795–834, 2010. doi: 10.1007/s10732-010-9126-2. URL https://www.scopus.com/inward/record.uri?eid=2-s2.0-78049529307{&}doi=10.1007{%}2Fs10732-010-9126-2{&}partnerID=40{&}md5=f933804350bfa3aa4c3984979b11576e.

F. Glover. Tabu search—part II. *ORSA Journal on Computing*, 2:4–32, 1990. doi: 10.1287/ijoc.2.1.4.

E. Guogis and A. Misevičius. Comparison of genetic programming, grammatical evolution and gene expression programming techniques. *Communications in Computer and Information Science*, 465:182–193, 2014. ISSN 18650929.

P. Hansen, N. Mladenovic, and J. Moreno. Variable Neighbourhood Search. *Inteligencia Artificial*, 7(19):77–92, 2003. ISSN 1137-3601. doi: 10.4114/ia.v7i19.717.

F. Herrera and M. Lozano. Adaptive genetic operators based on coevolution with fuzzy behaviors. *IEEE Transactions on Evolutionary Computation*, 5(2):149–165, 2001.

C. Huang, Y. Li, and X. Yao. A Survey of Automatic Parameter Tuning Methods for Metaheuristics. *IEEE Transactions on Evolutionary Computation*, 24(2):201–216, 2020. ISSN 19410026. doi: 10.1109/TEVC.2019.2921598.

D. Huang, T. T. Allen, W. I. Notz, and N. Zeng. Global optimization of stochastic black-box systems via sequential kriging meta-models. *Journal of global optimization*, 34(3):441–466, 2006.

F. Hutter, H. Hoos, K. Leyton-Brown, and T. Stützle. ParamILS: an Automatic Algorithm Configuration Framework. *Journal of Artificial Intelligence Research*, 36:267–306, 2009. ISSN 10769757. doi: 10.1613/jair.2808.

F. Hutter, H. H. Hoos, and K. Leyton-Brown. Sequential model-based optimization for general algorithm configuration. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 6683 LNCS:507–523, 2011. ISSN 03029743. doi: 10.1007/978-3-642-25566-3_40.

L. Ingber. Very fast simulated re-annealing. *Mathematical and computer modelling*, 12(8):967–973, 1989.

D. R. Jones, M. Schonlau, and W. J. Welch. Efficient global optimization of expensive black-box functions. *Journal of Global optimization*, 13(4):455–492, 1998.

L. Kaufman and F. Broeckx. An Algorithm for the Quadratic Assignment Problem Using Benders' Decomposition. *European Journal of Operational Research*, 2:207–211, feb 1978. doi: 10.1016/0377-2217(78)90095-4.

M. Kern. *Parameter adaptationn in heuristic search: a population based approach.* PhD thesis, University of Essex, 2006.

S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi. Optimization by simulated annealing. *Science*, 220 4598:671–80, 1983.

T. C. Koopmans and M. Beckmann. Assignment Problems and the Location of Economic Activities. *Econometrica*, 25(1):53–76, 1957. ISSN 00129682. doi: 10.2307/1907742. URL `http://dx.doi.org/10.2307/1907742`.

E. M. Loiola, N. M. M. de Abreu, P. O. Boaventura-Netto, P. M. Hahn, and T. Querido. A survey for the quadratic assignment problem. *European Journal of Operational Research*, 176(2):657–690, 2007a. ISSN 03772217. doi: 10.1016/j.ejor.2005.09.032.

E. M. Loiola, N. M. M. de Abreu, P. O. B. Netto, P. Hahn, and T. M. Querido. A survey for the quadratic assignment problem. *European Journal of Operational Research*, 176(2):657–690, 2007b. doi: 10.1016/j.ejor.2005.09.032. URL `http://dx.doi.org/10.1016/j.ejor.2005.09.032`.

H. R. Lourenço, O. C. Martin, and T. Stützle. Iterated local search. In *Handbook of metaheuristics*, pages 320–353. Springer, 2003.

R. J. Marshall, M. Johnston, and M. Zhang. A comparison between two evolutionary hyper-heuristics for combinatorial optimisation. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 8886:618–630, 2014. URL `https://www.scopus.com/inward/record.uri?eid=2-s2.0-84921419165{&}partnerID=40{&}md5=0c33f8ec9687c0740687125ce888e1a5`.

R. Martí and M. Laguna. Scatter Search : Diseño Básico y Estrategias Avanzadas. *Inteligencia Artificial, Revista Iberoamericana de Inteligencia Artificial.*, 7(19):1–8, 2003.

D. Merkle and M. Middendorf. Chapter 14 SWARM INTELLIGENCE. *Swarm Intelligence*, page 418, 2001.

D. Munera, D. Diaz, and S. Abreu. *Solving the quadratic assignment problem with cooperative parallel extremal optimization*, volume 9595. 2016. ISBN 9783319306971. doi: 10.1007/978-3-319-30698-8_17.

V. Nannen and A. E. Eiben. A method for parameter calibration and relevance estimation in evolutionary algorithms. In *Proceedings of the 8th annual conference on Genetic and evolutionary computation*, pages 183–190, 2006.

A. D. A. Neto and E. Martins. An adaptive multi-objective heuristic search for model-based testing. In *2018 11th International Conference on the Quality of Information and Communications Technology (QUATIC)*, pages 193–200. IEEE, 2018.

G. Ochoa, M. Hyde, T. Curtois, J. A. Vazquez-Rodriguez, J. Walker, M. Gendreau, G. Kendall, B. McCollum, A. J. Parkes, S. Petrovic, and E. K. Burke. HyFlex: A benchmark framework for cross-domain heuristic search. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 7245 LNCS:136–147, 2012. ISSN 03029743. doi: 10.1007/978-3-642-29124-1_12.

M. Oltean and C. Grosan. A Comparison of Several Linear Genetic Programming Techniques. *Complex Systems*, 14(4):285–313, 2004. ISSN 0891-2513. URL `http://www.cs.ubbcluj.ro/{~}cgrosan/030409{_}edited.pdf`.

G. Palubeckis. An Algorithm for Construction of Test Cases for the Quadratic Assignment Problem. *Informatica, Lith. Acad. Sci.*, 11(3):281–296, 2000.

R. S. Parpinelli, G. F. Plichoski, R. S. D. Silva, and P. H. Narloch. A review of techniques for online control of parameters in swarm intelligence and evolutionary computation algorithms. *International Journal of Bio-Inspired Computation*, 13(1): 1–20, 2019.

R. Pavón, F. Díaz, and V. Luzón. A model for parameter setting based on bayesian networks. *Engineering Applications of Artificial Intelligence*, 21(1):14–25, 2008.

R. Pavón, F. Díaz, R. Laza, and V. Luzón. Automatic parameter tuning with a bayesian case-based reasoning system. a case of study. *Expert Systems with Applications*, 36(2):3407–3420, 2009.

M. E. H. Pedersen. *Tuning & simplifying heuristical optimization*. PhD thesis, University of Southampton, 2010.

M. Prais and C. C. Ribeiro. Reactive grasp: An application to a matrix decomposition problem in tdma traffic assignment. *INFORMS Journal on Computing*, 12(3):164–176, 2000.

I. Rechenberg. *Evolutions strategie: Optimierung technischer Systeme nach Prinzipien der biologischen Evolution*, volume 15. Frommann-Holzboog, 1973. ISBN 9783772803741. URL https://books.google.de/books?id=iw2hcQAACAAJ.

M. G. C. Resende and J. L. Gonzáles Velarde. Greedy randomized adaptive search procedures (GRASP). *Encyclopedia of optimization*, 2(2):373–382, 2003. doi: doi: 10.1088/1475-7516/2007/08/005.

J. Ries and P. Beullens. A semi-automated design of instance-based fuzzy parameter tuning for metaheuristics based on decision tree induction. *Journal of the Operational Research Society*, 66(5):782–793, 2015. ISSN 1476-9360. doi: 10.1057/jors.2014.46. URL https://doi.org/10.1057/jors.2014.46.

J. Ries, P. Beullens, and D. Salt. Instance-specific multi-objective parameter tuning based on fuzzy logic. *European Journal of Operational Research*, 218(2):305–315, 2012.

P. Ross. Hyper-heuristics. In: Burke EK and Kendall G (eds). In *Search Methodologies: Introductory Tutorials in Optimization and Decision Support Techniques*, chapter 17, pages 529–556. 2005.

N. R. Sabar and G. Kendall. Population based Monte Carlo tree search hyper-heuristic for combinatorial optimization problems. *Information Sciences*, 314: 225–239, 2014. ISSN 00200255. doi: 10.1016/j.ins.2014.10.045. URL http://dx.doi.org/10.1016/j.ins.2014.10.045.

N. R. Sabar, M. Ayob, G. Kendall, and R. Qu. Grammatical evolution hyper-heuristic for combinatorial optimization problems. *IEEE Transactions on Evolutionary Computation*, 17(6):840–861, 2013. doi: 10.1109/TEVC.2013.2281527. URL https://www.scopus.com/inward/record.uri?eid=2-s2.0-84890355666{&}doi=10.1109{%}2FTEVC.2013.2281527{&}partnerID=40{&}md5=14c1d238e894047cc880701b79596104.

N. R. Sabar, M. Ayob, G. Kendall, and R. Qu. A dynamic multiarmed bandit-gene expression programming hyper-heuristic for combinatorial optimization problems. *IEEE Transactions on Cybernetics*, 45(2):217–228, 2014. doi: 10.1109/TCYB.2014.2323936. URL https://www.scopus.com/inward/record.uri?eid=2-s2.0-84921411064{&}doi=10.1109{%}2FTCYB.2014.2323936{&}partnerID=40{&}md5=ef107ae6eb93b86e81febc507bd8c118.

N. R. Sabar, M. Ayob, G. Kendall, and R. Qu. Automatic design of a hyper-heuristic framework with gene expression programming for combinatorial optimization problems. *IEEE Transactions on Evolutionary Computation*, 19(3):309–325, 2015. doi: 10.1109/TEVC.2014.2319051. URL https://www.scopus.com/inward/record.uri?eid=2-s2.0-84930941322{&}doi=10.1109{%}2FTEVC.2014.2319051{&}partnerID=40{&}md5=89e656599f5c419c976ff3a771292499.

J. Sacks, W. J. Welch, T. J. Mitchell, and H. P. Wynn. Design and analysis of computer experiments. *Statistical science*, pages 409–423, 1989.

S. Sahni and T. Gonzalez. P-Complete Approximation Problems. *Journal of the ACM*, 23(3):555–565, 1976. ISSN 00045411. doi: 10.1145/321958.321975.

M. Saifullah Hussin. *Stochastic Local Search Algorithms for Single and Bi-objective Quadratic Assignment Problems*. PhD thesis, Université de Bruxelles, 2016.

H.-P. Schwefel. Evolutionsstrategien für die numerische optimierung. In *Numerische Optimierung von Computer-Modellen mittels der Evolutionsstrategie*, pages 123–176. Springer, 1977.

M. Sevaux, K. Sörensen, and N. Pillay. Adaptive and multilevel metaheuristics. 06 2015.

K. Sim and E. Hart. An Improved Immune Inspired Hyper-heuristic for Combinatorial Optimisation Problems. In *Proceedings of the 2014 Annual Conference on Genetic and Evolutionary Computation*, GECCO '14, pages 121–128, New York, NY, USA, 2014. ACM. ISBN 978-1-4503-2662-9. doi: 10.1145/2576768.2598241. URL `http://doi.acm.org/10.1145/2576768.2598241`.

T. Stützle and M. López-Ibáñez. Automated design of metaheuristic algorithms. In *Handbook of metaheuristics*, pages 541–579. Springer, 2019.

N. Su, M. Zhang, and M. Johnston. A genetic programming based hyper-heuristic approach for combinatorial optimisation. *Genetic and Evolutionary Computation Conference, GECCO'11*, (October 2015):1299–1306, 2011. doi: 10.1145/2001576.2001752.

E. Taillard. Robust taboo search for the quadratic assignment problem. *Parallel Computing*, 17(4-5):443–455, 1991. ISSN 01678191. doi: 10.1016/S0167-8191(05)80147-4.

E.-G. Talbi. A taxonomy of hybrid metaheuristics. *Journal of Heuristics*, 8:541–564, 01 2002. doi: 10.1023/A:1016540724870.

A. Turky, N. R. Sabar, S. Dunstall, and A. Song. *Hyper-heuristic Based Local Search for Combinatorial Optimisation Problems*, volume 3339. Springer International Publishing, 2018. ISBN 978-3-540-24059-4. doi: 10.1007/b104336. URL `http://link.springer.com/10.1007/b104336`.

S. Voss, J. Born, and I. Santibanez-Koref. Looking ahead with the pilot method. *Annals of Operations Research*, 285-302:225–242, 2005.

D. H. Wolpert and W. G. Macready. No free lunch theorems for optimization. *IEEE Transactions on Evolutionary Computation*, 1(1):67–82, 1997. ISSN 1089778X. doi: 10.1109/4235.585893.

Z. Wu, Y. Yang, F. Bai, and J. Tian. Global optimality conditions and optimization methods for quadratic assignment problems. *Applied Mathematics and Computation*, 218(11):6214–6231, 2012. ISSN 00963003. doi: 10.1016/j.amc.2011.11.068.

M. Yagiura and T. Ibaraki. Local Search, 2002.

M. Zennaki and A. Ech-Cherif. A new machine learning based approach for tuning metaheuristics for the solution of hard combinatorial optimization problems. *Journal of Applied Sciences(Faisalabad)*, 10(18):1991–2000, 2010.

J. Zhang, W.-N. Chen, Z.-H. Zhan, W.-J. Yu, Y.-L. Li, N. Chen, and Q. Zhou. A survey on algorithm adaptation in evolutionary computation. *Frontiers of Electrical and Electronic Engineering*, 7(1):16–31, 2012.