UNIVERSIDADE DE SÃO PAULO

PROGRAMA DE PÓS-GRADUAÇÃO EM ENGENHARIA ELÉTRICA

VINICIUS RENAN DE CARVALHO

USING MULTI-AGENT SYSTEMS AND SOCIAL CHOICE THEORY TO DESIGN HYPER-HEURISTICS FOR MULTI-OBJECTIVE OPTIMIZATION PROBLEMS

São Paulo
2022

VINICIUS RENAN DE CARVALHO

USING MULTI-AGENT SYSTEMS AND SOCIAL CHOICE THEORY TO DESIGN HYPER-HEURISTICS FOR MULTI-OBJECTIVE OPTIMIZATION PROBLEMS

Corrected Version

Thesis submitted for the degree of Doctor in Science to the Escola Politécnica of Universidade de São Paulo.

São Paulo
2022

VINICIUS RENAN DE CARVALHO

Using multi-agent systems and social choice theory to design hyper-heuristics for multi-objective optimization problems

Corrected Version

Thesis submitted for the degree of Doctor in Science to the Escola Politécnica of Universidade de São Paulo.

Concentration field:

Computer Engineering

Advisor:

Prof. Dr. Jaime Simão Sichman

São Paulo
2022

Catalogação-na-publicação

de Carvalho, Vinicius Renan
Using multi-agent systems and social choice theory to design hyper heuristics for multi-objective optimization problems / V. R. de Carvalho -- versão corr. -- São Paulo, 2022.
125 p.

Tese (Doutorado) - Escola Politécnica da Universidade de São Paulo. Departamento de Engenharia de Computação e Sistemas Digitais.

1.Hyper-heuristicas 2.Sistemas Multi-Agentes 3.Otimização Multi-objetivo 4.Teoria da Escolha Social 5.Votação Baseadas em Agentes I.Universidade de São Paulo. Escola Politécnica. Departamento de Engenharia de Computação e Sistemas Digitais II.t.

"What does it mean... To be strong?"

Ippo Makunouchi

"When you gaze long into the abyss.
The abyss gazes also into you."

Friedrich Nietzsche

"Ansiava por ter conforto, dinheiro, um nome, pois só assim conseguiria matar aquela insuportável sensação de fracasso, de inferioridade."

Erico Verissimo.

"Não sou nada. Nunca serei nada. Não posso querer ser nada. À parte isso, tenho em mim todos os sonhos do mundo."

Álvaro de Campos

"Computers are like old Testament Gods; lots of rules and no mercy."

Joseph Campbell

"I wanna thank me for, for never quitting"

Snoop Dogg

"The saddest aspect of life right now is that science gathers knowledge faster than society gathers wisdom"

Isaac Asimov

"The universe is not required to be in perfect harmony with human ambition."

Carl Sagan

"If there is evil in this world, it lurks in the hearts of man."

Edward D. Morrison

"Homem, por que trabalhas com tanta fúria durante todas as horas de sol? — ouviria esta resposta singular: Para ganhar a vida. E no entanto a vida ali estava a se oferecer toda, numa gratuidade milagrosa."

Erico Verissimo

To Dayane Caldeira, who stayed at my side during good and bad moments, especially the bad ones.

ACKNOWLEDGEMENT

Being a Ph.D. candidate is not an easy task since it demands effort and patience from the student. However, being a student in pandemic times while fighting against a lymphoma made it even harder.

Fortunately, I have good people by my side, and the first one I would like to thank is Dayane Caldeira, my girlfriend to whom this thesis is dedicated to, by staying at my side "in sickness and in health" even we did not swear it in a church.

Thanks to my mother Natalina, who always is worried about my happiness and health. She has listened to my complaints and comments about life, the universe, and everything. Mom always told me to study because education is the best path for freedom.

To all my family, in special, Jessé de Carvalho, Leonardo Batista e José Carlos Batista.

To all nurses and hematologist physicians from Hospital Brigadeiro, especially to Gabriela Novaes, Elisangela Silva, Fernanda de Morais, Mauro Freitas and Ana Paula who always had heard about this Ph.D. and made my life easier when I was at the hospital taking chemotherapy without any visitor due to the pandemic times. Thanks, in memoriam, to Fabio Gargitter and Adalto Reis for the wonderful talks while we were together battling cancer at the same room.

I want to thank Jaime Simão Sichman for the guidance and patience during all those years and to all PCS members at the University of São Paulo, especially Felipe Leno, Igor Conrado Alves, Luis Ludescher, Arthur Gusmão, Rodrigo Bonini, Ricardo Jacomini, and Ruben Glatt.

To Michele Luz, Doris Moriyama, Anne Caroline, Roger Pellizzoni and Leonardo Ferreira for the great friendship we made in the UK, and to Thainá Mariani and Giovani Guizzo by the friendship we have since my master's degree at the Federal University of Parana.

To Marcio Pascal (magote.com) and Anna Brody (Carta Healthcare) for entrusting me nice and good work during these hard times. Special thanks to my work colleagues (at Carta Healthcare) Vitor Lobão and Guillherme Milleo, for being really good people, really nice friends.

Thanks To Ender Özcan for receiving me as an international student at the University of Nottingham for one year. and a special thanks to other ASAP/COL laboratory colleagues Valdivino Santiago, Simon (Peng Shi), Derya Deliktaş, Vivek Ramamoorthy, and Wenwen Li (also by providing MOHH-RILA and mIBEA code).

Finally, I gratefully acknowledge the funding agencies that made this research possible: Conselho Nacional de Desenvolvimento Científico e Tecnológico (CNPq), grant agreement no. 140974/2016-4, and to Coordenação de Aperfeiçoamento de Pessoal de Nível Superior (CAPES), Finance Code 001.

RESUMO

A maioria dos algoritmos mais eficazes e eficientes para otimização multi-objetivo são baseados em Computação Evolucionária. Entretanto, o ato de escolher o algoritmo mais apropriado para solucionar um dado problema não é trivial, e sempre requer diversas execuções, o que custa tempo. Hiper-heurísticas de seleção fazem parte de uma área de pesquisa emergente que investiga diversas técnicas para detectar a melhor heurística-de-baixo-nível enquanto o problema de otimização é resolvido. Por outro lado, agentes são componentes autônomos responsáveis por monitorar um ambiente e executar algumas ações de acordo com suas percepções. Neste contexto, técnicas baseadas em agentes mostram-se adequadas para o projeto de hiper-heurísticas. Existem diversas hiper-heurísticas propostas para controlar heurísticas-de-baixo-nível, mas apenas poucas são focadas em selecionar algoritmos evolucionários multi-objetivo. Este trabalho apresenta uma hiper-heurística baseada em agentes focada em escolher o melhor algoritmo evolucionário multi-objetivo. Baseado na Teoria da Escolha Social, o arcabouço proposto executa um procedimento de votação cooperativo, considerando um conjunto de eleitores, que votam baseados em um indicador de qualidade, para definir qual algoritmo deve gerar mais soluções ao longo da execução. Análises comparativas de desempenho foram realizadas empregando diversos problemas de otimização do mundo-real. Resultados mostraram que a abordagem proposta foi muito competitiva tanto quando comparada ao melhor algoritmo para cada problema como também quando comparada a outras hiper-heurísticas do estado-da-arte.

Palavras-chave: Hyper-heuristícas, Sistemas Multi-Agente, Otimização Multi-Objetivo, Teoria da Escolha Social, Votação Baseada em Agentes, Método de Copeland, Método de Contagem de Borda, Método de Kemeny-Young.

ABSTRACT

The majority of the most effective and efficient algorithms for multi-objective optimization are based on Evolutionary Computation. However, choosing the most appropriate algorithm to solve a certain problem is not trivial and often requires a time-consuming trial process. As an emerging area of research, hyper-heuristics investigates various techniques to detect the best low-level heuristic while the optimization problem is being solved. On the other hand, agents are autonomous component responsible for watching an environment and perform some actions according to their perceptions. In this context, agent-based techniques seem suitable for the design of hyper-heuristics. There are several hyper-heuristics proposed for controlling low-level heuristics, but only a few of them are focused on selecting multi-objective optimization algorithms (MOEA). This work presents an agent-based hyper-heuristic for choosing the best multi-objective evolutionary algorithm. Based on Social Choice Theory, the proposed framework performs a cooperative voting procedure, considering a set of quality indicator voters, to define which algorithm should generate more offspring along to the execution. Comparative performance analysis was performed across several benchmark functions and real-world problems. Results showed the proposed approach was very competitive both against the best MOEA for each given problem and against state-of-art hyper-heuristics.

Keywords: Hyper-heuristics, Multi-Agent Systems, Multi-objective Optimization, Social Choice Theory, Agent-Based Voting, Copeland Method, Borda Count Method, Kemeny-Young Method.

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS

AE	Algorithm Effort
ASP	Algorithm Selection Problem
DE	Differential Evolution
DTLZ	Deb, Thiele, Laumanns and Zitzler's Benchmark
EA	Evolutionary Algorithm
GD	Generational Distance
GDE3	Generalized Differential Evolution 3
HH	Hyper-heuristic
HHCF	Maashi's Choice Function Hyper-Heuristic
HHLA	MOHH-RILA instance with no LLH removal during the initialization
HHRL	MOHH-RILA instance with LLH removal during the initialization
HR	Hypervolume Ratio
IBEA	Indicator Based Evolutionary Algorithm
IGD	Inverted Generational Distance
LLH	Low-Level Heuristic
MAS	Multi-Agent System
mIBEA	Modified Indicator Based Evolutionary Algorithm
MOABHH	Multi-Objective Agent-Based Hyper-Heuristic
MOEA	Multi-Objective Evolutionary Algorithm
MOHH-RILA	Learning Automata-based Multi-Objective Hyper-Heuristic with a ranking scheme initialization
MOP	Multi-Objective Optimization Problem
moTSP	Multi-objective Travel Salesman Problem

NSGA-II	Non-Dominated Sorting Genetic Algorithm II
PF	Pareto Front
RNI	Ratio of Non-Dominated Solution
SPEA2	Strength Pareto Evolutionary Algorithm 2
UD	Uniform Distribution
USP	Universidade de São Paulo
WFG	Walking Fish Group's benchmark
ZDT	Zitzler, Deb and Thiele's Benchmark

LIST OF SYMBOLS

$\mathscr{A}$	Algorithm Space
$\mathfrak{active}$	Function that generates $\mathscr{LLH}_{active}$
dv	Number of decision variables
E	External population
$\mathscr{E}$	Performance Evaluators Space
$\mathscr{EO}$	Election Outcome
$f(s)$	Objective function of a solution s
$f(\mathsf{X})$	Objective function of a decision variable set $\mathscr{X}$
g	Max generations
m	Number of quality indicators
N	Population size
n	Number of candidates
no	Number of objectives
ND	Non-Dominated Set
$\overline{N}$	Archive size
$\mathscr{LLH}$	Set of Low-Level Heuristics
$\mathscr{LLH}_{active}$	Set of active Low-Level Heuristics
O	Offspring population
$\mathscr{P}$	Problem Space
P	Population of solutions
PF_k	Known Pareto Front
PF_t	True Pareto Front
$\mathscr{PP}$	Participation Set

$\mathfrak{part}$	Function that generates $\mathscr{P}\mathscr{P}$
$\mathscr{R}$	Performance Evaluations
$\mathscr{R}\mathscr{P}$	Set of resulting sub-populations
S	Set of solutions
s	Solution for an optimization problem
$\mathscr{S}c$	Criterion of Selection
$\mathscr{V}$	Voters set
$\mathscr{V}\mathscr{M}$	Voting method set
$\mathscr{V}\mathscr{P}$	Voting preferences
$\mathfrak{vote}$	Function that maps $\mathscr{R}\mathscr{P}$ to $\mathscr{V}$
X	Decision variable set

CONTENTS

1 INTRODUCTION

Meta-heuristics are algorithms, solution methods that orchestrate an interaction between local improvement procedures (heuristics) and higher-level strategies to create a process capable of escaping from local optima and performing a robust search of a solution space (GENDREAU; POTVIN, 2010). Thus, usually, heuristics specialize in solving problems for one particular domain, while meta-heuristics are more generic and adaptive in several domains. Due to meta-heuristics generality, this kind of algorithm is widely used to solve complex optimization problems in industry and services in areas ranging from finance to production management and engineering (BOUSSAID; LEPAGNOT; SIARRY, 2013).

One of the most used meta-heuristics class is Evolutionary Algorithms (EAs) (COELLO, 2007). EAs are meta-heuristics which employ Darwin's theory of the survival of the fittest as their inspiration. This kind of algorithm generates solutions using the crossover and mutation heuristic operators. It employs an objective function to choose which individuals (solutions for the problem) will compose the next population.

In order to employ an evolutionary algorithm to find solutions for a given problem, an objective function is needed in order to evaluate the quality of generated solutions. An objective function has some input parameters and output, which tell us how good a given solution is. Each output is called an objective value. Thus, these problems can be classified according to the number of objectives in their objective function as *mono-objective* problems (for one objective) and *multi-objective* problems (MOPs) for more than one.

Multi-Objective Evolutionary Algorithms (MOEAs) have been successfully applied to solve MOPs (BOUSSAID; LEPAGNOT; SIARRY, 2013). Since MOEAs are evolutionary algorithms, they allow a flexible representation of the problem solutions and the use of problem-specific heuristics. Due to the general and abstract characteristics of MOEAs, researchers have proposed several algorithms to cope with MOPs (LI et al., 2015a).

In MOPs, no single algorithm can outperform the others in all problems, and algorithms perform differently depending on the problem characteristics. In this context, over the years, some techniques have been proposed in order to tackle the *Algorithm Selection Problem*, among them one called *Hyper-heuristics*.

Hyper-heuristics are high-level heuristics that can be used to reduce the difficulty of selecting the most suitable *Low-Level Heuristic* (LLH) for a given problem. A LLH can be either a heuristic operator such as crossover and mutation, or a meta-heuristic, such as evolutionary algorithms. Moreover, hyper-heuristics may or may not employ learning techniques. When applied, learning may occur online, when it is performed along with the search, or offline, by employing a training set beforehand.

Most research in the field of Hyper-heuristic has been limited to treating low-level heuristic selection, and the majority of research has been limited to treat mono-objective optimization problems (MAASHI; ÖZCAN; KENDALL, 2014). Moreover, there are even fewer works that focus on multi-objective algorithm selection.

The present work proposes a generic framework designed as a selection hyper-heuristics for choosing the most suitable multi-objective evolutionary algorithm (MOEA), responsible for solving multi-objective optimization problems. This online approach selects MOEAs, and for this reason, it is considered as an online learning hyper-heuristic.

The framework concept consists of applying Social Choice Theory techniques (LITTLE, 1952), in special voting methods, to design a multi-objective agent-based hyper-heuristic. Voting methods are employed due to their ability to resolve disagreements among different quality indicators, which are responsible for evaluating MOEAs performance.

In the proposed approach, MOEAs are defined as candidates to be voted in one election, and multi-objective quality indicators are defined as voters. Thus, voters are responsible for evaluating and sorting their algorithm preferences according to their quality evaluation. Finally, using the election outcome, the proposed hyper-heuristic defines how each algorithm will act. This approach can be adapted to work with different meta-heuristics, quality measures and voting methods. Figure 1 presents how the present work is categorized. Blue lines mean relation, black lines mean no relation.

Employing an election composed of candidates and voters makes the approach better designed as a distributed system, allowing the voters to vote simultaneously. In this thesis, MOEAs are competing against each other for gaining computational time, but they are also cooperating by aiming to solve a common optimization problem. For these reasons, multi-agent systems was considered the most suitable strategy for the design. Thus, candidates (algorithms) and voters (quality measures) can be better designed as agents.

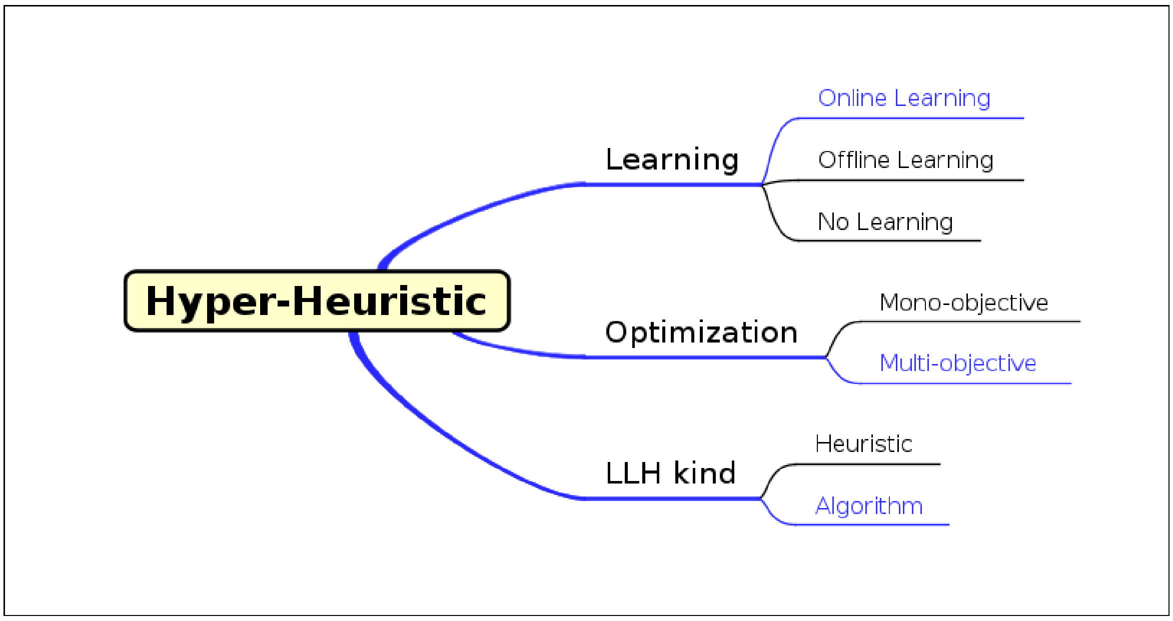


Figure 1: Present work related areas

1.1 Motivation

Choosing a heuristic or meta-heuristic for a given problem is not a trivial task, because there are many different options. Thus, to define which algorithm to use, it is necessary to perform a tuning method that consists of executing all meta-heuristics and evaluate their results to find the most suitable algorithm. A tuning method usually takes a long time and cannot be easily performed.

Since meta-heuristics are employed by researchers from different areas, approaches focused on reducing the effort on choosing a meta-heuristics are interesting. However, just a few works focus on multi-objective algorithm selection.

This present work can be classified as an on-line hyper-heuristic. This kind of algorithm needs no training, and it can find the best algorithm while solving an optimization problem. Sometimes, this kind of hyper-heuristic can also outperform meta-heuristic in overall results, which means, find better solutions than all single meta-heuristics across a set of different problems. This makes online hyper-heuristics even more interesting because it allows researchers to solve their problems with less effort.

1.2 Objective

The main goal of this research is to define a multi-agent election-based hyper-heuristic framework, which reduces the user effort on choosing multi-objective evolutionary algorithms. Thus, the user can just specify his problem and leave the task of choosing the best MOEA for the proposed hyper-heuristic. In order to illustrate the effectiveness of the proposed hyper-heuristic, some experiments were performed to answer the following questions:

RQ1 How does the proposed hyper-heuristic's performance change when applied to different MOPs? This question is necessary to evaluate the results obtained by the proposed HH when compared with meta-heuristics and other HHs when considering several MOPs.

RQ2 How does the hyper-heuristic performance change when using different voting methods? This question is necessary to analyze the proposed HH instances, that means, when the proposed HH is instantiated using different voting methods.

In order to pursue the main goal, the following sub-goals were needed:

1. Define an election-based multi-agent online hyper-heuristic concentrated on multi-objective problems. This hyper-heuristic must contain a set of algorithms, quality indicators, and a predefined voting method;
2. Explore the behavior of agents (candidates and voters) in different voting methods considering several benchmark problems;
3. Evaluate the proposed hyper-heuristic according to different quality indicators in order to find if the proposed approach finds competitive results against tuned algorithms.
4. Evaluate the proposed hyper-heuristic performance on real-world problems. Benchmarks usually provide information about their problems, such as worse and best solutions that can be found. In real-world problems, this information is unknown. Thus, for better comparisons, real-world problems are employed.

On the other hand, this work did not have as goals:

1. This work did not aim to create a new hybrid meta-heuristics;
2. The approach was not applied to mono-objective meta-heuristics;
3. This is a hyper-heuristic and not a meta-learning approach. Thus, no machine learning algorithms were considered a candidate in elections, for example, linear and logistic regression, Support vector machines, clustering, and deep learning techniques.
4. The proposed approach is not an automatic parameter tuning method, which is responsible for online selecting parameters for a given algorithm, but an online hyper-heuristic. Thus, it can be compared just with other online hyper-heuristics;
5. No new voting method was proposed, just existing ones were considered.

1.3 Methodology

First, a literature review on multi-objective hyper-heuristics was carried out to identify the current literature scenario and existing gaps. This review also included multi-agent approaches, but focused on the best approaches, even if they were not agent-based.

After a review, a model elaboration was started. This model had been designed considering knowledge from Multi-Agent Systems, Social Choice Theory, and Evolutionary Computation areas. As a result, an election-based hyper-heuristic framework was defined and implemented.

Experiments were performed in order to evaluate the proposed approach. In a first experiment (Section 7.2), originally published in (CARVALHO; SICHMAN, 2017), the proposed hyper-heuristic was instantiated using three different voting methods described in Chapter 4 in order to solve the benchmark WFG (Section 2.5.1) for two and three objectives. The second experiment (Section 7.3), originally published in (CARVALHO; SICHMAN, 2018b; CARVALHO et al., 2020), also instantiates the proposed HH using the three voting methods, but in this case, to find solutions for some real-world problems (Section 2.5.2). In both Experiment I and Experiment II, the HH performance was compared against the meta-heuristics outcome, which means full instances of the meta-heuristics, which are also employed as LLH. Finally, a third experiment was performed (Section 7.4) considering 18 real-world problems, the three HH instances (for each voting method), and the results were compared to three other state-of-art hyper-heuristic. These results were originally published in (CARVALHO; ÖZCAN; SICHMAN, 2021).

1.4 Contributions

To our current knowledge, this was the first work in the literature that combined multi-objective optimization, online hyper-heuristics, multi-agent systems and voting methods. Moreover, this work differs from state-of-art MOP hyper-heuristics in how low-level-heuristics run: instead of running all in sequence, by adopting a MAS approach all LLH may be executed in parallel. So, solutions can be optimized at the same time by the different LLHs. This is an interesting aspect due to the fact the LLH performance is evaluated in the same momentum, instead of considering LLH performance in different search times.

In order to evaluate the proposed framework, some instances were created combining different algorithms and voting procedures. In the beginning, just *benchmarks* were employed and the proposed approach was just compared against the best MOEA available in the LLH set. Along with this research, we employed real-world optimization problems and started to compare the results against state-of-art hyper-heuristics. In the end, several real-world problems were employed in order to evaluate how this hyper-heuristic can diminish the effort of decision-makers and increase search performance.

The results have shown the proposed approach overcomes current state-of-art approaches. The combination of multi-agent concepts and preferences summarization (Social Choice Theory) can, in fact, improve the selection of Low-Level Heuristics in online hyper-heuristics.

1.5 Text Organization

The text is divided into three parts and composed of eight chapters and one appendix. Part I is composed of Chapters 2, 3 and 4; while Part II is composed of Chapters 5 and 6. Finally, Part III is composed of Chapter 7 and 8. All the chapters are described next:

- **Chapter 2 - Multi-objective optimization and evolutionary computation** covers basic concepts of heuristics, meta-heuristics, multi-objective problems, multi-objective algorithms and quality indicators;

- **Chapter 3 - Algorithm Selection Problem** introduces the Algorithm Selection Problem and Hyper-heuristics;
- **Chapter 4 - Multi-Agent Systems and Social Choice Theory** introduces concepts related to agents, and covers basic ideas of social choice and voting methods, needed to fully understand this thesis;
- **Chapter 5 - Multi-agent Election-Based Hyper-Heuristic** - presents and details this Thesis proposal;
- **Chapter 6 - Related Work** - presents the related work;
- **Chapter 7 - Results** - shows the preliminary results obtained until now;
- **Chapter 8 Conclusion** - presents conclusion and Further Work;
- **Appendix A: List of Publications** - details the publications related to this thesis.

PART I

BACKGROUND

2 MULTI-OBJECTIVE OPTIMIZATION AND EVOLUTIONARY COMPUTATION

This present chapter introduces concepts related to Evolutionary Computation and Multi-objective Optimization. First, Section 2.1 succinctly introduces optimization concepts. Section 2.2 presents heuristic, meta-heuristic and evolutionary algorithms. Then, in Section 2.3, Multi-objective evolutionary algorithms are discussed and some of the main algorithms are discussed. In Section 2.4, some quality indicators, that are necessary for performance evaluation in the context of multi-objective optimization, are presented. Finally, in Section 2.5, some multi-objective problems, that can be applied to test our approach, are described.

2.1 Optimization problems

An optimization problem consists of finding extremum values (minimal or maximum) for a mathematical function, commonly named as *fitness* or *objective* function. For this purpose, a set of decision variables X $= \{x_1, ..., x_d v\}$ (with size dv) have to be considered. In this context, a *solution* is a value assignment for each variable of this set of decision variables. Thus, to evaluate how well a solution solves the optimization problem, we just need to calculate the objective function $f($X$) = f(x_1, x_2, x_3, ..., x_n)$. This objective function can either be minimization ($\min f($X$)$) or maximization ($\max f($X$)$). Both decision variables and objective functions can be continuous (X $\in \mathbb{R}$) or discrete (X $\in \mathbb{Z}$). Thus, optimization problems can also be classified as discrete or continuous.

Many real-world problems need the specification of multiple objective functions in order to evaluate solutions using different criteria at the same time. For example, suppose the case of buying a car considering, at the same time, price and fuel consumption. In this case, the objective function has a set of outputs with size no. In these problems, solutions should optimize different and often conflicting criteria (COELLO, 2007). So, in these problems, we aim to find the set of optimal trade-off solutions known as the Pareto optimal set (BRADSTREET et al., 2007). These problems are known as multi-objective problems (MOPs).

In Pareto dominance, a certain solution s_a in the decision space of a MOP is superior to another solution s_b if and only if $f(s_a)$ is at least as good as $f(s_b)$ in terms of all the objectives and strictly better than $f(s_b)$ in terms of at least one single objective. Solution s_a is also said to strictly dominate solution s_b (ADRA, 2007).

The Pareto front is a subset of the search space which contains all Pareto optimal solutions. Hence, there are a set of solutions, which do not dominate one another. Figure 2 presents a two-objective minimization problem. Here we can see The Pareto front (in blue) and a set of dominated solutions (in red). Two non-dominated solutions (s_1 and s_2) are highlighted, both belong to the Pareto Front. Solution s_1 prioritizes objective 2 while s_2 objective 1. There are 23 other solutions that are also not-dominated (in blue). Thus, the final unique solution has to be selected by a decision-maker responsible for evaluating the trade-off among the 25 found solutions.

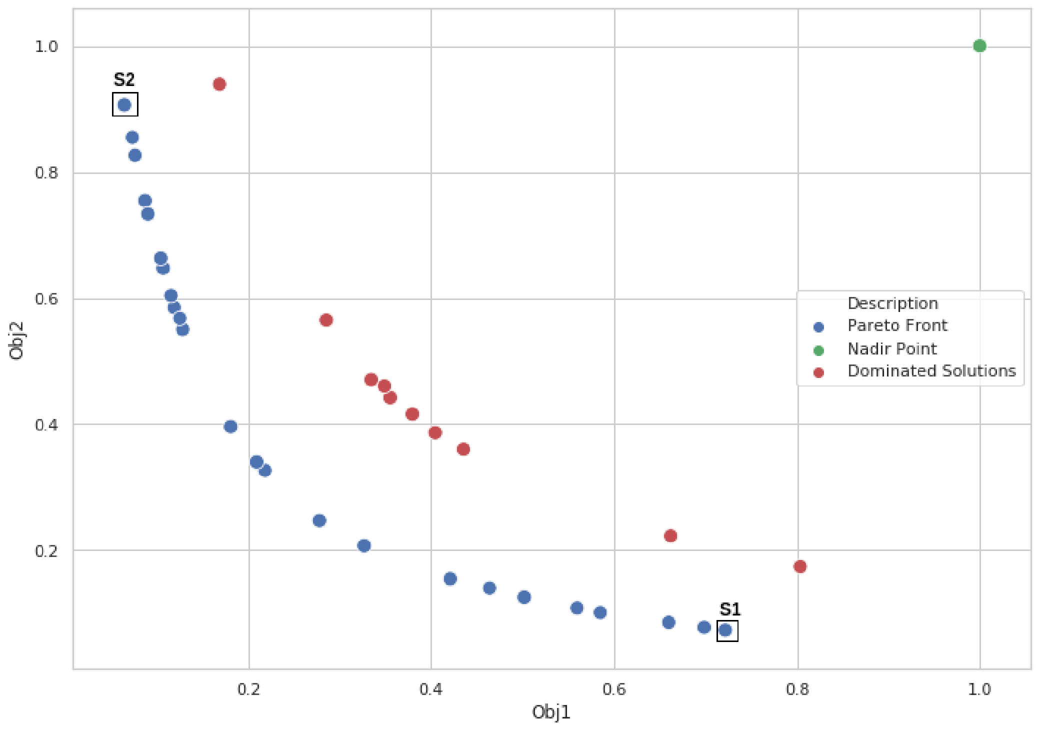


Figure 2: Two objectives Pareto Front Minimization Example

Represented in green in Figure 2 , the *nadir point* represents the worse possible solution for the problem. The *nadir point* is important since it is used as the reference point by some quality indicators, enabling to compare different algorithms performance on solving MOPs.

Furthermore, Fronts are used as a reference by some quality indicators. Benchmark problems have reference sets composed of mathematically found solutions. These solutions are the best possible, and there is no way to excel them. In this case, we name this front as the True Pareto Front (PF_t). On the other hand, real-world problems do not have PF_t. However, one can generate a front to be used as a reference. In this case, we generate a front composed of all non-dominated solutions, found across all the experiments, by all the studied algorithms for a given problem. Thus, there is no way for a studied algorithm to surpass solutions from this generated set. This set is named as Known Pareto Front (PF_k).

Evolutionary Algorithms are the most used algorithm to solve optimization problems (COELLO, 2007), and are presented next.

2.2 Evolutionary Algorithms

Heuristics are criteria, methods, or principles for deciding which among several alternatives courses of action promises to be the most effective in order to achieve some goal (PEARL, 1984). Heuristics do not guarantee optimal solutions; in fact, they do not guarantee any solutions at all; all that can be said for a useful heuristic is that it offers solutions that are good enough most of the time (FEIGENBAUM; FELDMAN et al., 1963).

Meta-heuristic, in turn, can be defined as an iterative generation process that guides a subordinate heuristic by combining intelligently different concepts for exploring and exploiting the search space in order to find efficiently near-optimal solutions (OSMAN; LAPORTE, 1996).

Usually, heuristics are specialized in solving problems for one particular domain, while meta-heuristics are more generic and adaptive in several domains.

Meta-heuristics can be classified into two groups: Single-Solution Based or Population-Based ones. In the single-solution approach, a single initial solution is generated and updated along with the search. Thus, just the best-found solution is stored. Simulated Annealing (AARTS; KORST, 1989) and GRASP (FEO; RESENDE, 1995) are examples of single-solution meta-heuristics.

Population-based meta-heuristics generate a population of solutions at the beginning of the search, and each iteration, update the population with newly generated solutions. The most studied population-based methods are related to Evolutionary Computation (EC) and Swarm Intelligence (SI) (BOUSSAID; LEPAGNOT; SIARRY, 2013).

Evolutionary Computation is the general term for several optimization algorithms that are inspired by the Darwinian principles of nature's capability to evolve living beings well adapted to their environment (BOUSSAID; LEPAGNOT; SIARRY, 2013). These algorithms are also called as Evolutionary Algorithms (EA), and they all share a common underlying idea of simulating the evolution of individual (or solution) structures via processes of selection, reproduction, and mutation, thereby producing better solutions (BOUSSAID; LEPAGNOT; SIARRY, 2013).

In order to discover which are the best solutions generated by evolutionary algorithms, we need a definition of an objective function. Thus, along with the execution, the fittest set is selected according to its objective value.

Algorithm 1 presents a generic Evolutionary Algorithm. In the beginning, the population is filled with random solutions (Line 3). Solutions are then evaluated according to the predefined objective function (Line 4). While a termination criterion is not reached (usually a max number of iterations), parents are selected (Line 6) and used by a heuristic to generate new solutions (Line 7). New solutions are evaluated according to the objective function (Line 8). Finally, offspring and parent solutions compete to survive and compose the next generation population (Line 9).

Algorithm 1: Generic Evolutionary Algorithm

```
1  Input: Optimization Problem, Termination Condition
2  begin
3      Initialize the population with random solutions;
4      Evaluate solutions according to the objective function;
5      while a termination condition is not satisfied do
6          Select parents;
7          Generate new solutions using some heuristic;
8          Evaluate new solutions according to the objective function;
9          Select solutions to compose the next generation;
10     end
11 end
```

In the literature, some algorithms implemented the concept of an evolutionary algorithm. That is the case of Genetic Algorithm (GOLDBERG; HOLLAND, 1988) and the Evolutionary Programming (KOZA, 1992), which mainly differs in how solutions are represented: genetic algorithms usually consider fixed-size arrays for the decision variables set, while evolutionary programming considers data structures like trees. These algorithms focused on mono-objective optimization, which means, one single value to represent the quality of a given solution. However, several real-world problems consider more than one objective value in order to evaluate individual quality properly. In this scenario, Multi-objective Evolutionary Algorithms (MOEAs) are able to find good solutions for this kind of problem (COELLO, 2007). At next, we present the distinguishing features of MOEAs.

2.3 Multi-objective Evolutionary Algorithms

MOPs are tackled today using Evolutionary Algorithms by engineers, computer scientists, biologists, and operations researchers alike (COELLO, 2007). Multi-objective Evolutionary Algorithms (MOEAs) are extensions of EAs for multi-objective problems that usually apply the concepts of Pareto dominance (ADRA, 2007) to create different strategies to evolve and diversify of the solutions.

The evolutionary computation community has been particularly active in this area and a vast number of evolutionary multi-objective algorithms have been proposed. Many of these algorithms are genetic algorithms that differ from each other, mainly in the way that solutions are ranked at every iteration (VÁZQUEZ-RODRÍGUEZ; PETROVIC, 2012).

At next, we present some of the most important MOEAs in the literature. Figure 3 presents a set of solutions (numbered from 1 to 14). In this example, offspring O and parent population P are already joined. Thus, this figure will be used to detail how the following MOEAs select surviving solutions.

2.3.1 NSGA-II

The main characteristic of the Non-Dominated Sorting Genetic Algorithm (NSGA-II) (DEB et al., 2002) is the application of a robust elitism mechanism, where at each generation, a non-dominated sorting is conducted and a domination rank is attributed to each solution. The NSGA-II builds a population of competing solutions, ranks and sorts each solution according to non-domination level, applies evolutionary operations to create a new pool of offspring, and then combines the parents and offspring before partitioning the new combined pool into fronts (COELLO, 2007).

Through the non-dominated-sort, the population is sorted by creating several fronts, where

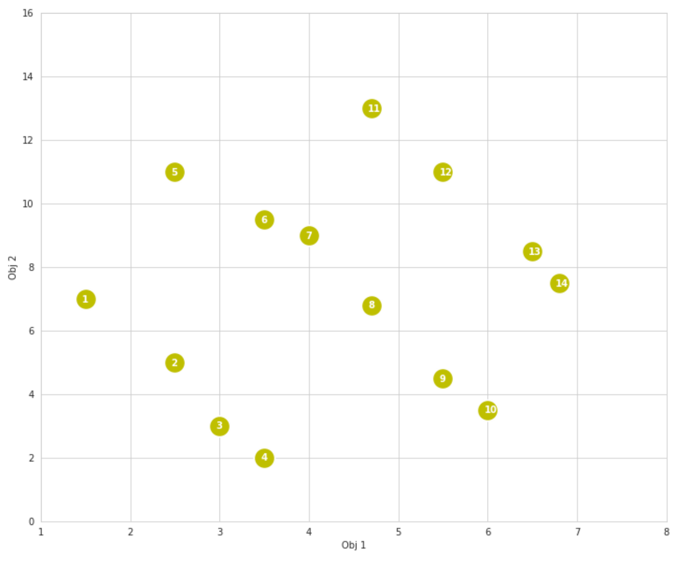


Figure 3: Solutions used as example

solutions in the first front are non-dominated regarding all solutions, and solutions from the second front are dominated only by the solutions of the first front, and so on. Then, the fronts are sequentially added to the next population. If by including a front, the size of the population is surpassed, then the solutions in this front are ranked according to their crowding distances.

The crowding distance is a diversity estimator extensively applied to evolutionary multi-objective algorithms to promote diversity. It estimates the density of solutions surrounding a particular point and only the best solutions from this last front are used to fill the population.

The new population is used to generate a new main population starting a new cycle of the algorithm. In the end, after the stop criterion, the algorithm returns the population as a result.

Algorithm 2 and Figure 4 present NSGA-II algorithm, which has as inputs the population size N, the objective function to be optimized $f(s)$, and the max number of generations g. At Lines 6 and 12, $rank$ values are assigned, these ranking tells how much dominated a solution is. At Line 13, subsets are created taking into consideration $rank$ values, and for each subset, the Crowding Distance is calculated (Line 14). This is necessary for sorting all subsets according to higher Crowding distance values (Line 15). So, at Line 16, a new population P_{t+1} is created by copying all complete subset until no more full subset can be added. At Line 18, in case $|P_{t+1}| < N$, last subset solutions (the one which could not be added), are selected according to Crowding Distance values and added to population P_{t+1} until $|P_{t+1}| = N$.

Algorithm 2: NSGA-II algorithm, adapted from (COELLO, 2007)

```
Input: $N$, $f(s)$, $g$
begin
    Initialize Population $P$;
    Generate Random Population - size $N$;
    Evaluate Objective values;
    Assign $rank$ (level) based on Pareto dominance - Sort;
    Generate offspring population $O$;
        Parent Selection;
        Recombination and mutation;
    for $i \leftarrow 1$ to $g$ do
        for each $s \in P \cup O$ do
            Assign $rank$ (level) based on Pareto dominance - Sort;
            Generate sets of nondominated vectors along with the search space;
            Assign Crowding Distance values;
            Sort subset solutions according to Crowding Distance values;
            Loop (inside) by adding solutions to next generation starting from the first
              front until $N$ individuals;
        end
        Select points (elitist) on the lower front (with lower rank) and are outside a
          crowding distance;
        Create next generation $O$;
            Parent Selection;
            Recombination and mutation;
    end
end
```

Figure 5 presents how NSGAII selects surviving solutions through its elitism method. We have 14 solutions (from Figure 3) competing for survival considering a max population size $N = 7$. So, first NSGA-II split solutions into subsets according to the *Dominance Level* (Figure 5a). Then, *Crowding-Distance* values are assigned (Figure 5b). These values are 0 for solutions inside a front for solutions containing the best values for an objective. For example, for the subset 0, solutions 1 and 4 receive 0, for the subset 1, solutions 5 and 10 also receive 0. Since NSGA-II considers non-dominance as the first criterion, solutions 1 to 4 are selected. Then we need to select three more solutions to complete the population, assuming we have $N = 7$. Reference solutions (5 and 10) are then selected. At this point, we need to select just one more solution. Hence, the one with higher *Crowding distance* is selected: solution 8. Thus, the surviving solutions for $N = 7$ are $\{1, 2, 3, 4, 5, 8, 10\}$, represented in blue in

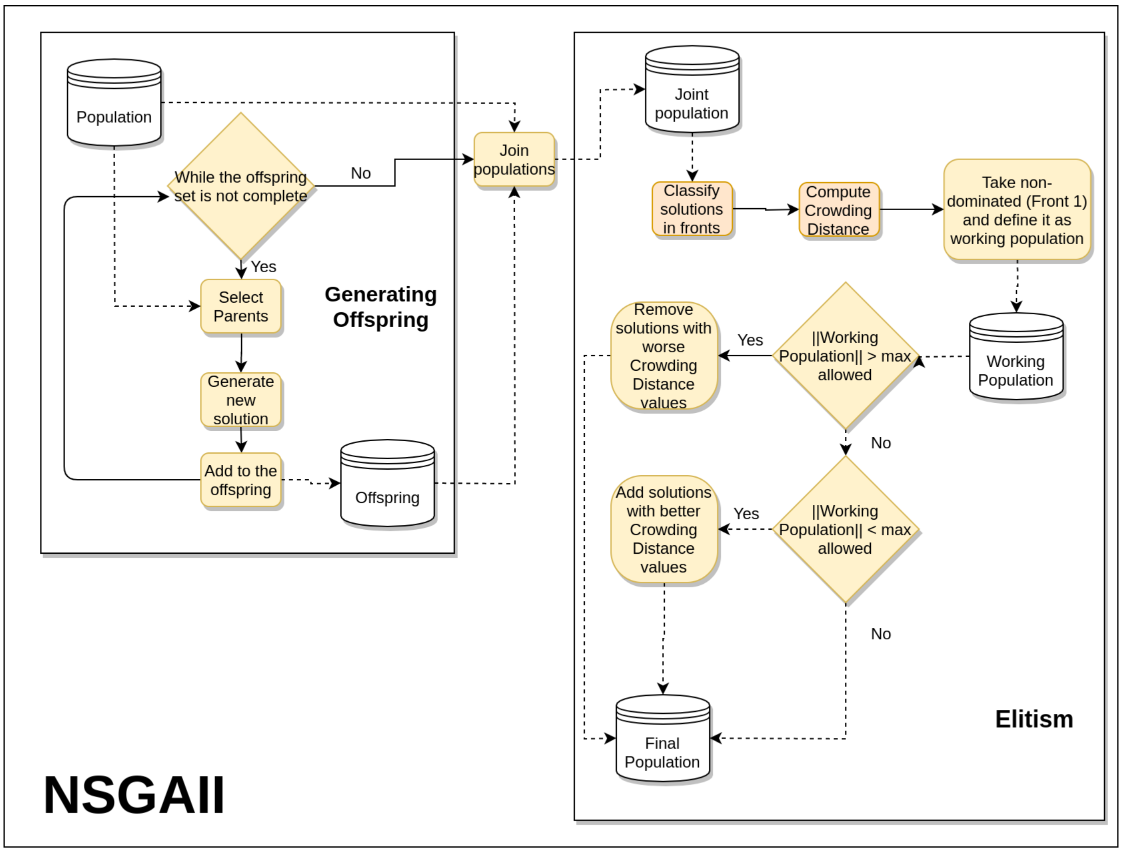
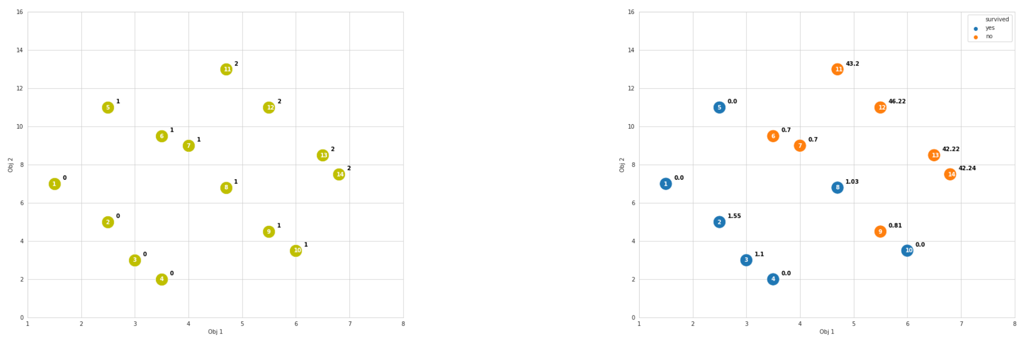

Population
While the offspring set is not complete
No
Join populations
Yes
Select Parents
Generating Offspring
Generate new solution
Add to the offspring
Offspring
Joint population
Classify solutions in fronts
Compute Crowding Distance
Take non-dominated (Front 1) and define it as working population
Working Population
||Working Population|| > max allowed
Yes
Remove solutions with worse Crowding Distance values
No
||Working Population|| < max allowed
Yes
Add solutions with better Crowding Distance values
No
Final Population
Elitism
NSGAII

Figure 4: NSGAII Example

Figure 5b. Another example would be if we had $N = 3$. In this case, solutions 1 and 4 would be selected. Then, solutions 2 and 3 would be compared, and solution 2 would be selected due to its higher *Crowding Distance* value, leading to $\{1, 4, 2\}$ as surviving solutions.



(a) Dominance Ranking (b) Crowding Distance

Figure 5: NSGA-II surviving selection

2.3.2 SPEA2

Strength Pareto Evolutionary Algorithm 2 (ZITZLER; LAUMANNS; THIELE, 2001) has two main differences in relation to NSGA-II: the way of ranking the solutions and the use of elitism through an external population named *archive*. SPEA2 is an algorithm that introduces the *Strength* value for selecting new solutions for the new population. This algorithm incorporates a fine-grained fitness assignment strategy, which considers for each solution the number of solutions that it dominates and that it is dominated by.

As in the NSGA-II, a joined population is created, but here the solutions are selected both from the population and from the archive. This joined population can be used either in the crossover and mutation operators to generate the offspring. The non-dominated solutions are used to fill the archive, and as the maximum size of the archive is determined by a parameter ($\overline{N}$), two situations can occur. In the first case, the number of non-dominated solutions is smaller than the archive size; in this case, the archive is filled with dominated solutions. In the second situation, if the number of non-dominated solutions is greater than the maximum archive size, then an operation to eliminate solutions is applied. Only individuals belonging to the archive survive for the next generation.

The fitness strategy is calculated according to Equation 2.1. First all solution *strength values* are calculated. A *strength value* for a given solution s is the number of solutions which dominate s. With all solutions *strength value*, we can calculate the $RawFitness$ value, which is the *strength values* sum of all solutions which dominate s.

$$Fitness_s = RawFitness_s + \frac{1}{(\text{distance}(s,k) + 2)}, \tag{2.1}$$

Equation 2.1 also use a Euclidean Distance between solution s to the k-th population solution, given by $k = \sqrt{N + \overline{N}}$.

Algorithm 3 and Figure6 present SPEA2 algorithm, which has as input the population size N, the archive size $\overline{N}$, the max number of generations g, and the objective function $f(s)$.

After initialization, SPEA2 begins calculating all solutions *fitness*, according to Equation 2.1, at Line 6. In the following, solution with lower *fitness* are copied to the archive (Line 7). So, in the case of $|E| > \overline{N}$, then an elimination operator is performed by calculating neighbors distances and removing the nearest neighbor. In case of $|E| < \overline{N}$, the archive

Algorithm 3: SPEA2 algorithm, adapted from (COELLO, 2007)

```
1  Input: $N$, $\overline{N}$, $g$, $f(s)$
2  begin
3      Initialize Population $P$;
4      Create empty archive $E$;
5      for $i \leftarrow 1$ to $g$ do
6          Compute $fitness$ of each individual in $P$ and $E$;
7          $E \leftarrow$ Non-dominatet set from $P \cup E$;
8          if Size of $E$ is bigger than $\overline{N}$ then
9              Use the truncation operator to remove elements from E when the capacity
                of the archive has been extended;
10         end
11         else if Size of $E$ is lower than $\overline{N}$ then
12             Use dominated individuals $P$ to fill $E$;
13         end
14         Perform parent selection;
15         Apply crossover and mutation to generate $P$;
16     end
17 end
```

receives dominated solutions from P. SPEA2 continues selecting parents, and performing crossover and mutation to generate a new offspring in P.

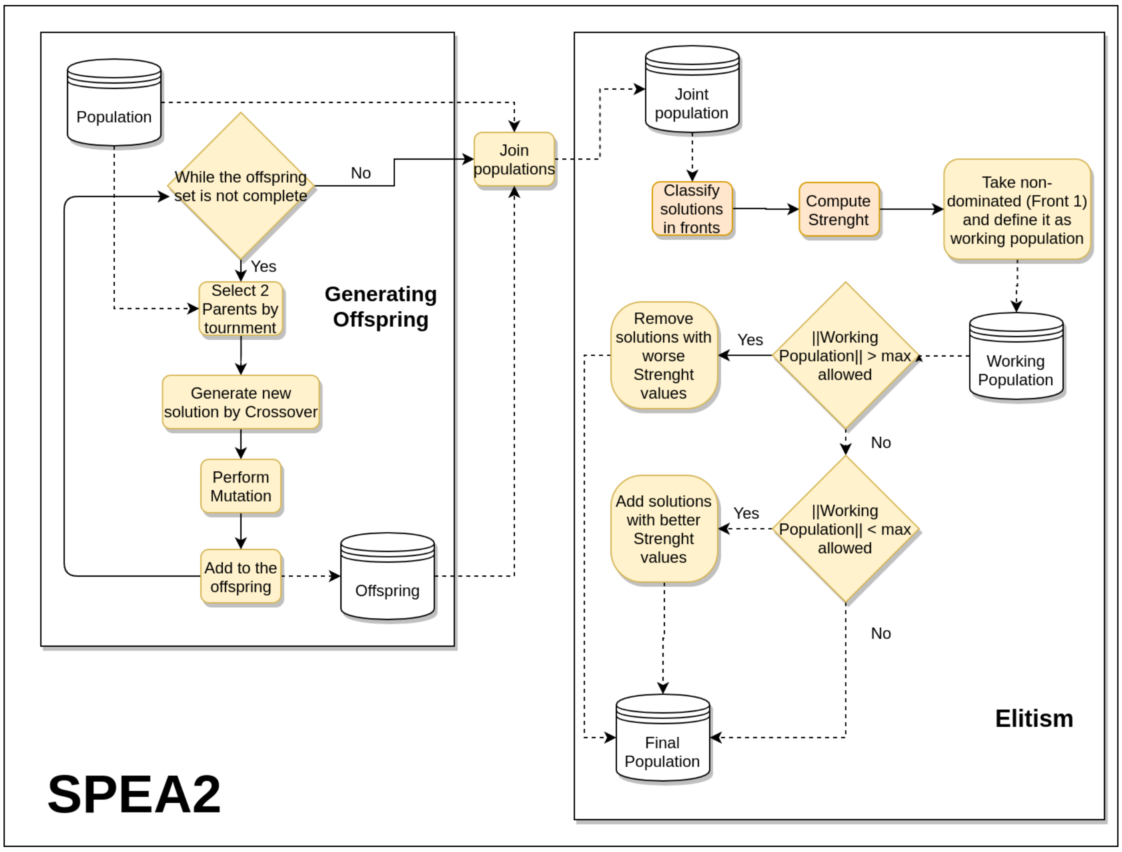

Population
While the offspring set is not complete
No
Yes
Select 2 Parents by tournment
Generating Offspring
Generate new solution by Crossover
Perform Mutation
Add to the offspring
Offspring
Join populations
Joint population
Classify solutions in fronts
Compute Strenght
Take non-dominated (Front 1) and define it as working population
Working Population
||Working Population|| > max allowed
Yes
Remove solutions with worse Strenght values
No
||Working Population|| < max allowed
Yes
Add solutions with better Strenght values
No
Final Population
Elitism
SPEA2

Figure 6: SPEA2 Flowchart

Figure 7 presents SPEA2 Fitness calculation and surviving selection, and also considers solutions from Figure 3. Since SPEA2 also uses *Dominance level* then Figure 5a is also applicable here as the first step. Then SPEA2 uses this information to calculate its Fitness. Thus, considering $N = \overline{N} = 7$, SPEA2 would select solutions $\{1, 2, 3, 4, 5, 9, 10\}$, represented in blue in Figure 7, since they have smaller fitness values. Solutions $\{11, 12, 13, 14\}$ weren't even considered, and for that reason, they don't have fitness calculated. Considering $N = \overline{N} = 3$, SPEA2 would also select solutions $\{1, 2, 4\}$.

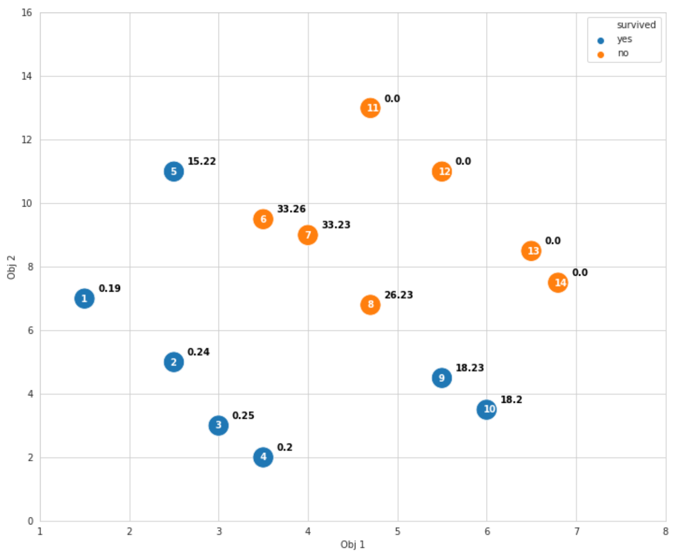


Figure 7: SPEA2 surviving selection

2.3.3 IBEA

The main idea of the Indicator-Based Evolutionary Algorithm (ZITZLER; KÜNZLI, 2004) is to use a quality indicator (Section 2.4) to select surviving solutions. This quality indicator is used to calculate the contribution of a solution; this means how much the quality increases if a given solution is kept in the population. Thus, the algorithm can compare different contributions to add to the new population the solutions which contribute more. One of the most known indicators used with IBEA is Hypervolume (ZITZLER; THIELE, 1999) due to its capacity to evaluate the convergence and diversity of the search process at the same time (ZITZLER; THIELE, 1999). More details about this indicator can be found in Section 2.4.2.

Algorithm 4 and Figure 8 show IBEA flow. It starts by creating a random population

(Line 3), an empty archive, and evaluating the objective values of population solutions (Line 4). Then, it repeats the following process until the stop criterion is satisfied: the parents are selected (Line 6) to be used in the crossover and mutation operations to generate the offspring (Line 7). After the reproduction step, the offspring are added to the population (Line 8). Then the quality indicator contribution is calculated (Line 9). So, while the size of the population exceeds, the worst individual (evaluated by the selected indicator) is removed from the population, and the population fitness is recalculated (Line 10). Finally, the archive receives the set containing the surviving solutions and the population is set as empty (Line 11). When the population reaches the size, a new cycle of the algorithm starts. In the end, IBEA returns a set of non-dominated solutions found.

Algorithm 4: IBEA algorithm

```
1  Input: $N$, $g$, $f(s)$
2  begin
3      Initialize Population $P$ and the archive $E$;
4      Evaluate Objective values;
5      for $i \leftarrow 1$ to $g$ do
6          Parent Selection using $E$;
7          Recombination and mutation using $P$;
8          $JoinedSet \leftarrow P \cup E$;
9          Calculate indicator contribution for each solution in $JoinedSet$;
10         Remove solutions with lower contribution until $|JoinedSet| = N$;
11         $E \leftarrow JoinedSet$ and $P \leftarrow \emptyset$;
12     end
13 end
```

Once more, Figure 9 presents how IBEA selects surviving solutions (in blue) of the same example used in this section. In this example, the Hypervolume contribution is calculated for all of the solutions considering as the reference the point $(0, 0)$ as reference. So solutions with smaller values contribute more. Thus, solutions $\{1, 2, 3, 4, 5, 8, 9\}$ are selected for $N = \overline{N} = 7$. With $N = \overline{N} = 3$ solutions $\{1, 3, 4\}$ would be selected.

2.3.4 mIBEA

The Modified Indicator-Based Evolutionary Algorithm (Li et al., 2017), based on IBEA, also employs Hypervolume as the quality indicator. Different from its predecessor, which considers all solutions in $E \cup P$ to select solutions to compose P' based on the quality indicator contribution, this algorithm uses only non-dominated solutions from the union set, and then

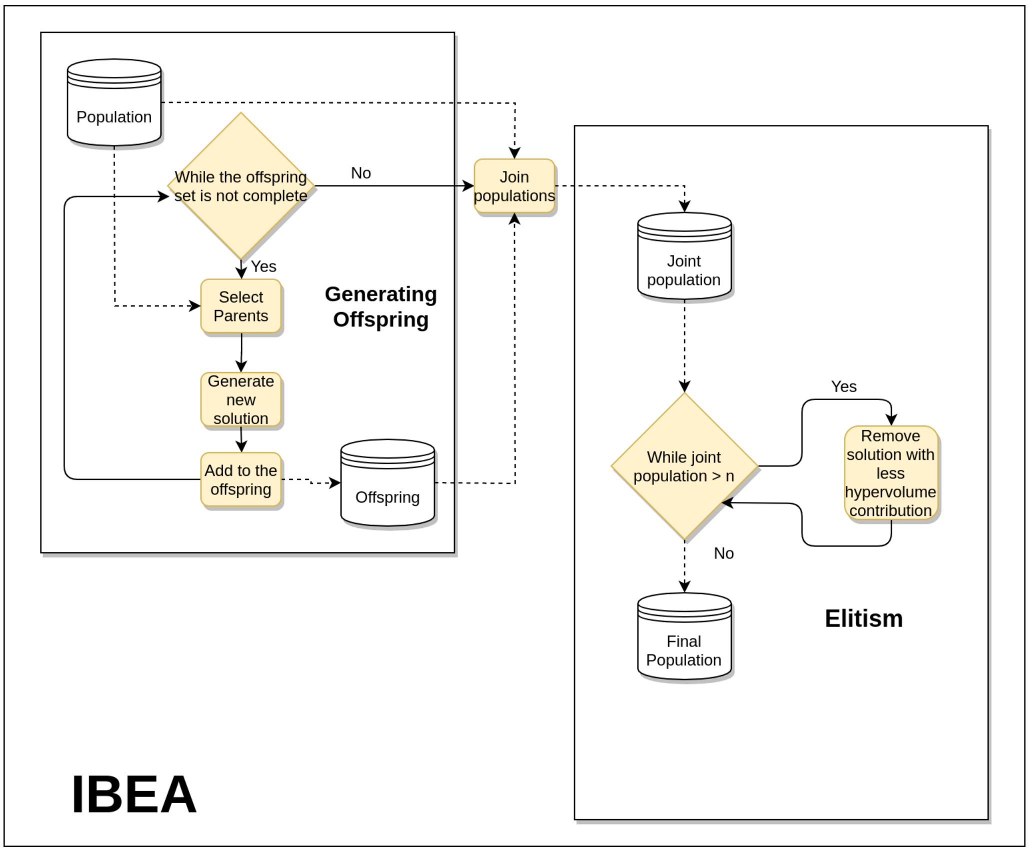


Figure 8: IBEA Flowchart

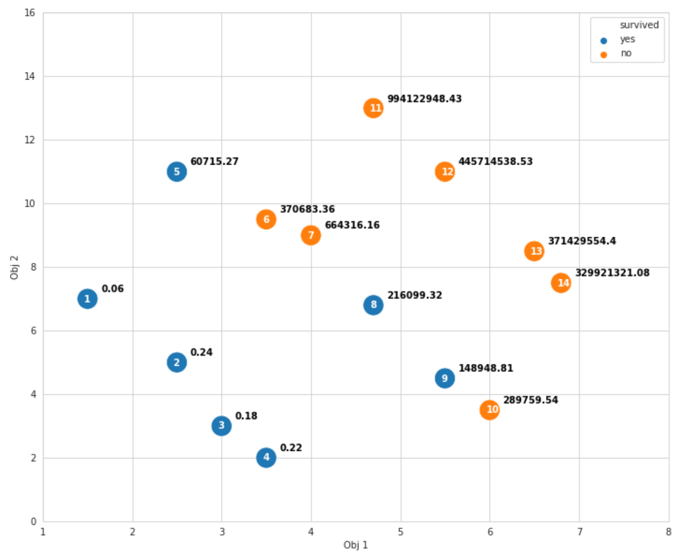


Figure 9: IBEA surviving selection

selects the ones which contribute more. This algorithm works in the same way as IBEA after this. This modification improves the algorithm convergence and removes solutions with high-quality indicator contributions, which are far away from the Pareto Front.

Algorithm 5 and Figure 10 show mIBEA flow. It starts by creating a random population (Line 3), an empty archive, and evaluating the objective values of population solutions (Line 4). Then, it repeats the following process until the stop criterion is satisfied: the parents are selected (Line 6) to be used in the crossover and mutation operations to generate the offspring (Line 7). After the reproduction step, the offspring are added to the population (Line 8). So the main difference between IBEA and mIBEA occurs, the algorithm removes all non-dominated solutions from the population (Line 9). Then the quality indicator contribution is calculated (Line 10). So, while the size of the population exceeds, the worst individual (evaluated by the selected indicator) is removed from the population, and the population fitness is recalculated (Line 11). Finally, the archive receives the set containing the surviving solutions and the population is set as empty (Line 12). When the population reaches the size, a new cycle of the algorithm starts. In the end, IBEA returns a set of non-dominated solutions found.

Algorithm 5: mIBEA algorithm

```
1  Input: $N$, $g$, $f(s)$
2  begin
3      Initialize Population $P$ and the archive $E$;
4      Evaluate Objective values;
5      for $i \leftarrow 1$ to $g$ do
6          Parent Selection using $E$;
7          Recombination and mutation using $P$;
8          $JoinedSet \leftarrow P \cup E$;
9          $NonDominated \leftarrow$ Remove dominated solutions from $JoinedSet$;
10         Calculate indicator contribution for each solution in $NonDominated$;
11         Remove solutions with lower contribution until $|NonDominated| = N$;
12         $E \leftarrow NonDominated$ and $P \leftarrow \emptyset$;
13     end
14 end
```

Using the same example of the section again, mIBEA would select just solutions $\{1, 2, 3, 4\}$ for $N = \overline{N} = 7$ and ignore other solutions, since it first removes dominated solutions. For $N = \overline{N} = 3$, four non-dominated solutions would compete, and solutions which smaller Fitness values would be selected: in this case, mIBEA would select $\{1, 3, 4\}$, the same that would be selected by IBEA. This algorithm, which considers *Pareto Dominance* as a criterion, uses more the quality indicator contribution (in this case, the Hypervolume contribution) when most of

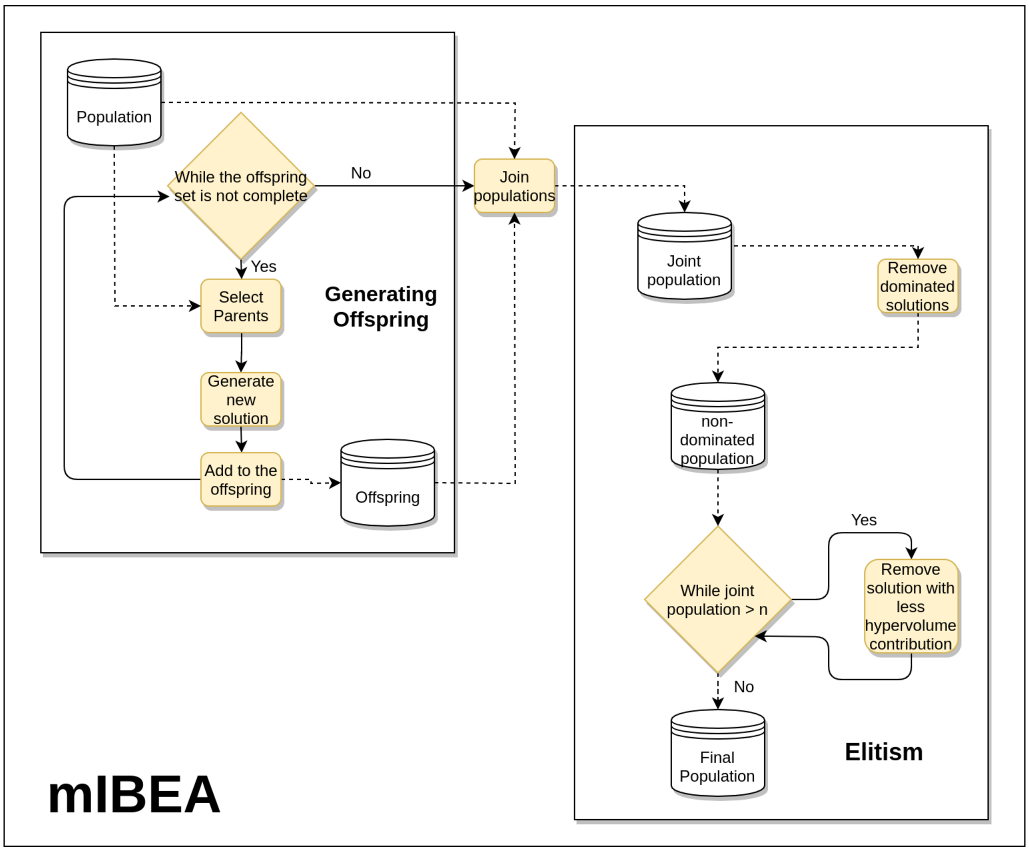


Figure 10: mIBEA Flowchart

the solutions are non-dominated.

2.3.5 GDE3

Generalized Differential Evolution 3 (KUKKONEN; LAMPINEN, 2005), differently from previously presented algorithms, doesn't employ a crossover operator to generate new solutions. This algorithm instead uses the differential evolution operator (STORN; PRICE, 1997).

Differential Evolution (DE) is a relatively new EA and it has been gaining popularity during previous years (KUKKONEN; LAMPINEN, 2005)

Different from the crossover operator, DE generates new solutions by combining more than three different solutions. In most cases, a DE is just applied on continuous optimization problems, because DE generates offspring parameters by calculating weighted differences among solutions parameters.

Table 1 presents the most used differential evolution equations in literature. These equations are performed using every solution parameter. In these equations, $s1$, $s2$, $s3$, $s4$ and $s5$

are independent solutions, randomly selected, different of the current solution sc. sb is the best solution in the current population for DE heuristic. F, F_1 and F_2 are the DE scaling factors. In this table, s_{dv} represents the decision variable set of a given solution s.

Table 1: Differential Evolution types

Name	Equation
DE/rand/1	$sc_{dv} \leftarrow s1_{dv} + F * (s2_{dv} - s3_{dv})$
DE/rand/2	$sc_{dv} \leftarrow s1_{dv} + F_1 * (s2_{dv} - s3_{dv}) + F_2 * (s4_{dv} - s5_{dv})$
DE/best/1	$sc_{dv} \leftarrow sb_{dv} + F * (s2_{dv} - s3_{dv})$
DE/best/2	$sc_{dv} \leftarrow sb_{dv} + F_1 * (s1_{dv} - s2_{dv}) + F_2 * (s3_{dv} - s4_{dv})$
DE/current-to-best/1	$sc_{dv} \leftarrow sc_{dv} + F_1 * (sb_{dv} - sc_{dv}) + F_2 * (s2_{dv} - s3_{dv})$

Algorithm 6 presents the GDE3 algorithm. First, the initial population is generated and evaluated (Lines 3 and 4). Then, until the max generations are reached, for every solution $s \in population$ (Line 7), the algorithm selects parent solutions (Line 8), find the best solution in the population (Line 9) (if necessary), then generates and adds the new solution s'_c to the offspring population (Line 10). After generating N new solutions, the algorithm evaluates the offspring population (line 13) and compare all parent with respective offspring solutions according to Pareto dominance to generate an auxiliary population (Line 14). The auxiliary population is submitted to a Crowding distance selection (Line 15), whether his size is bigger than N. Finally, the auxiliary population becomes the current population (Line 16).

Considering the selection of surviving solutions, GDE3 is similar to NSGA-II, since both algorithms implements Pareto-Dominance and Crowding-Distance. The difference between these algorithms lies in how they generate solutions: while NSGAII uses crossover, GDE3 employs differential evolution instead. Using the same example of this section, GDE3 would select the same solutions chosen by NSGA-II, and therefore the same results obtained in Figure 5 would be obtained by GDE3.

2.3.6 Discussion

Two kinds of evolutionary algorithms were presented in this section: Genetic algorithms such as NSGA-II, SPEA2, IBEA and mIBEA; and one algorithm based on differential evolution (GDE3).

Among these algorithms, NSGA-II is undoubtedly the most known in the literature and the fastest (in computational time) of the four genetic algorithms. mIBEA is the second

Algorithm 6: GDE3 algorithm

```
1  Input: $N$, $g$, $f(s)$
2  begin
3      Initialize Population $P$;
4      Evaluate Objective values;
5      for $i \leftarrow 1$ to $g$ do
6          $O \leftarrow \emptyset$;
7          for Each $s_c \in population$ do
8              Select from $P$ some distinct parents $(s_1, s_2, s_3, s_4, s_5)$ different of $s_c$;
9              Find the best solution in population;
10             Generate a new solution $s'_c$ with the Differential Evolution operator;
11             $O \leftarrow O + \{s'_c\}$;
12         end
13         Evaluate $O$ objective values;
14         Compare parent $(P)$ and offspring $(O)$ solutions according to Pareto
             Dominance to generate an auxiliary population $aux$;
15         Perform the Crowding Distance selection into $aux$;
16         $P \leftarrow aux$;
17     end
18 end
```

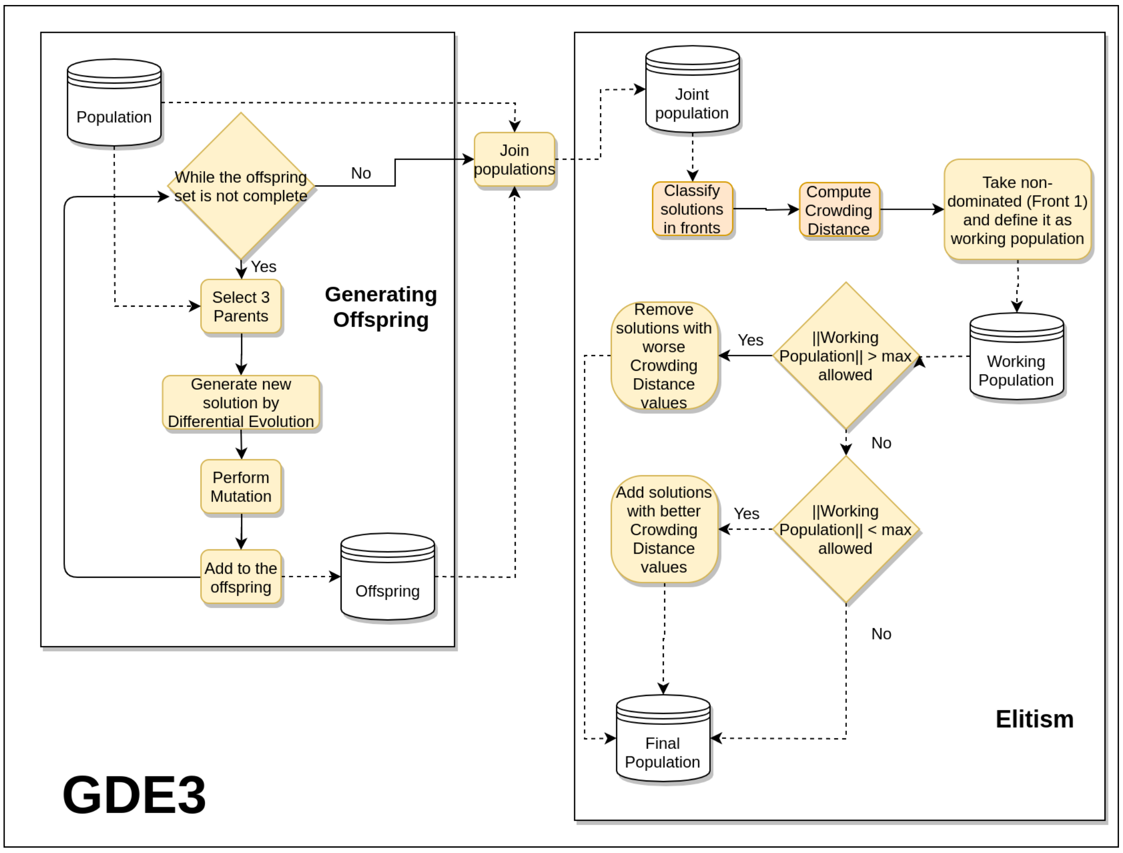

Population
While the offspring set is not complete
No
Yes
Select 3 Parents
Generating Offspring
Generate new solution by Differential Evolution
Perform Mutation
Add to the offspring
Offspring
Join populations
Joint population
Classify solutions in fronts
Compute Crowding Distance
Take non-dominated (Front 1) and define it as working population
Working Population
||Working Population|| > max allowed
Yes
Remove solutions with worse Crowding Distance values
No
||Working Population|| < max allowed
Yes
Add solutions with better Crowding Distance values
No
Final Population
Elitism
GDE3

Figure 11: GDE3 Flowchart

fattest. IBEA and SPEA2 are the two slowest. In case we consider more than five objectives functions IBEA becomes the slowest, followed by mIBEA due to the high computational cost of calculating multi-dimensional hypervolumes. GDE3 is only here which can just be applied for continuous optimization, due to the fact of using differential evolution.

The No Free Lunch Theorem indicates that search algorithms such as MOEAs are not individually robust over all problems by definition (COELLO, 2007). For example, in (BRADSTREET et al., 2007), the authors compared NSGA-II and SPEA2 using the WFG benchmark (HUBAND et al., 2006) and found NSGA-II better than SPEA2 on two-objective instances. However, increasing the number of objectives makes SPEA2 outperform NSGA-II. In (YEN; HE, 2014) the authors compared NSGA-II, SPEA2, IBEA, using ZDT (ZITZLER; DEB; THIELE, 2000) and DTLZ (DEB, K. et al, 2005) benchmarks, and no algorithm outperformed all others in all problems. The same happened in (Li et al., 2017) considering mIBEA and these other algorithms.

2.4 Quality Indicators

Due to the fact that MOPs having a set of solutions (Pareto Front) instead of a single solution, direct comparisons between algorithms results cannot be directly performed. In order to allow algorithm comparison, some quality indicators were proposed to assess a MOEA performance in solving multi-objective optimization problems. In this session, some of them are presented. Most of them employ a returning set of solutions (S) in order to evaluate a MOEA quality.

2.4.1 RNI

The ratio of non-dominated solutions (TAN; LEE; KHOR, 2002) evaluates the percent of non-dominated solutions ($ND(S)$) in the population (S), as shown in Equation 2.2. Higher RNI values are better than lower ones.

$$RNI(S) = \frac{|ND(S)|}{|S|} \tag{2.2}$$

2.4.2 Hypervolume

The hypervolume (ZITZLER; THIELE, 1999) of a non-dominated solution set S is the size of the part of objective space that is dominated collectively by the solutions in S (WHILE; BRADSTREET; BARONE, 2012). Thus, the hypervolume indicator computes the area (or volume when more than two objectives are employed) in the search space (ZITZLER; THIELE, 1999). Equation 2.3 presents how to calculate this indicator, where v_i is the volume. Higher hypervolumes are preferred to lower ones when the reference point is the *Nadir point*.

$$HV(S) = volume(\cup_{i=1}^{|S|}) \tag{2.3}$$

Different from other quality indicators, there are several ways to calculate the Hypervolume in the literature, for example (WHILE; BRADSTREET; BARONE, 2012) and (ZITZLER; THIELE, 1999). This is necessary due to the fact this calculation is a costly task to perform when several solutions are considered. So these implementations consider mathematical properties in order to speed up the task. In fact, all of them aim to obtain a single value that represents an area/volume/hypervolume as showed in Figure 12. In this figure, the four non-dominated solutions used in the example in Section 2.3 are considered. So the union area among the solutions $\{1, 2, 3, 4\}$ and the *Nadir point* at the position $(6, 12)$ is calculated.

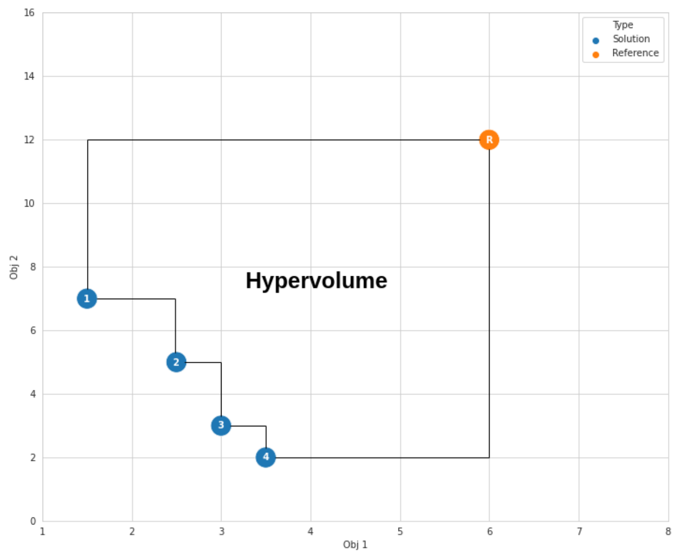


Figure 12: Hypervolume

2.4.3 R2

The R2 indicator (HANSEN et al., 1998) employs the standard weighted Tchebycheff function in order to calculate the mean difference in utility values. This indicator simultaneously evaluates the convergence and diversity of an approximation set (KNOWLES; CORNE, 2002). Lower $R2$ values are preferred to lower ones.

Equation 2.4 presents R2, S is the solution set, W the weight vector, and Z^* the reference point set. Inside the keys is the standard weighted Tchebycheff function, which calculates the max absolute difference value between an objective in the ideal point Z_j^* and a given solution objective value s_j.

$$R2(S, W, Z^*) = \frac{1}{|S|} \sum_{w \in W} \min_{s \in S} \left\{ \ \max_{1 \leq j \leq no} w_j |Z_j^* - s_j| \ \right\} \tag{2.4}$$

2.4.4 UD

The Uniform distribution of non-dominated population evaluates how distributed are the solutions along with the search space. The distribution should be as uniform as possible to achieve consistent gaps among neighboring individuals in the population (TAN; LEE; KHOR, 2002). This quality indicator is calculated according to Equation 2.5.

$$UD(S) = \frac{1}{1 + S_{nc}} \tag{2.5}$$

where S_{nc} is the standard deviation of the niche count of the overall set of $NonDominated(S)$ (Equation 2.6).

$$S_{nc}(ND) = \sqrt{\frac{\sum_i^{|ND|}(nc(nd_i) - \overline{nc}(ND))^2}{|ND| - 1}} \tag{2.6}$$

where $|ND|$ is the size of the non-dominated set ND of the population S; $nc(nd_i)$ is the niche count of i^{th} a solution;

$$nc(nd_i) = \sum_{j, j \neq i}^{|ND|} f(i, j), f(i, j) = \left\{ \begin{array}{l} 1 \text{ if } dist(i, j) < \sigma_{share} \\ 0 \text{ otherwise} \end{array} \right\} \tag{2.7}$$

and $\overline{nc}(ND)$ is the mean value of $nc(nd_i)$

$$\overline{nc}(ND) = \frac{\sum_i^{|ND|} nc(nd_i)}{|ND|} \tag{2.8}$$

where dist(i, j) is the distance between individual i and j in the objective domain.

2.4.5 GD

The Generational Distance (SRINIVAS; DEB, 1994) corresponds to the sum of the Euclidean distances between the outcome population of solutions S and the solutions in Pareto Front P. In general, it is impossible to find all solutions on a continuous Pareto Front (JIANG et al., 2014); in this case, the Pareto Front is previously known. Equation 2.9 presents how to calculate the GD indicator, where d is the Euclidean distance from $i \in S$ to $q \in P$. Lower GD values are better than higher ones.

$$GD(S, P) = \frac{(\sum_{i=1}^{|S|} d_i^q)^{\frac{1}{q}}}{|S|} \tag{2.9}$$

Usually, q is defined as 2 in the literature.

2.4.6 IGD and IGD+

The Inverted Generational Distance (ZITZLER et al., 2003) is very similar to the GD. Instead of calculating the Euclidean distance from S to P, as GD does, this indicator calculates the Euclidean distance from P to S. Equation 2.10 presents how to calculate the IGD indicator. Similarly to the GD indicator, lower values are preferred to higher ones.

$$IGD(S, P) = \frac{(\sum_{i=1}^{|P|} d_i^q)^{\frac{1}{q}}}{|P|} \tag{2.10}$$

The Inverted Generational Distance Plus (ISHIBUCHI et al., 2015) is the low computational cost version of its predecessor, which also considers the dominance relationship between the Pareto-optimal reference set and a given population of solutions. For this purpose, when a solution set P is better than another solution set S in terms of the Pareto dominance relation, S is never evaluated as being better than P by the IGD+ indicator. In this case, S can be evaluated as being better than P by the IGD indicator (ISHIBUCHI et al., 2019).

2.4.7 Spread

The Spread (SRINIVAS; DEB, 1994), or Δ metric, evaluates the distribution of non-dominated solutions over the Pareto Front (MAASHI; ÖZCAN; KENDALL, 2014). Equation 2.11 presents how to calculate the Spread indicator, where d_i is the Euclidean distance between solutions in sequence, $\overline{d}$ is the distance mean of d_f, d_l are the minimum Euclidean distances from solutions in S to the extreme (bounding) solutions of P (JIANG et al., 2014). Lower spreads values are preferred to lower ones.

$$\Delta(S,P) = \frac{d_f + d_l + \sum_{i=1}^{|S|-1} |d_i - \overline{d}|}{d_f + d_l + (|S|-1)\overline{d}} \tag{2.11}$$

2.4.8 HR

The Hyper-area Ratio (HR) (VELDHUIZEN, 1999) employs the hypervolume of a solution set A divided by the hypervolume value of a Reference Front B. Higher values are preferred to lower ones.

$$HV(S,P) = \frac{HV(A)}{HV(B)} \tag{2.12}$$

2.4.9 ER

Pareto Dominance Indicator (ER) (GOH; TAN, 2009) considers the intersection of the solution between two given sets A and B, which can be provided by different algorithms or used to compare a solution set S with a Pareto Front PF, and then $|S \cap PF|$. Equation 2.13 presents ER, where the size of the intersection is compared with the size of B. Higher values are preferred to lower ones.

$$ER(A,B) = \frac{|A \cap B|}{|B|} \tag{2.13}$$

2.4.10 AE

The Algorithm Effort quality indicator can be defined as the ratio of the total number of function evaluations N_{eval} over a fixed period of simulation time T_{run} (TAN; LEE; KHOR, 2002). This indicator is interesting to evaluate how fast a MOEA can generate solutions.

$$AE = \frac{T_{run}}{N_{eval}} \tag{2.14}$$

2.4.11 Discussion

When evaluating the performance of a MOEA, there are two main goals to pursue: (1) *convergence*: closeness of the provided non-dominated solution set to the Pareto optimal front and (2) *divergence*: diversity of the obtained solution set (with a good distribution) along the Pareto optimal front (ELARBI et al., 2017). Table 2 summarize the quality indicators presented. In this table, an additional column shows whether PF (PF_{true} or PF_{known}) is necessary (specifying the previous best-known solution set) and other shows if this quality indicator has to be maximized or minimized.

Table 2: Quality indicators comparison

Quality Indicator	Convergence	Divergence	Require PF	Maximizing/Minimizing
AE				minimizing
GD	✓		✓	minimizing
IGD	✓	✓	✓	minimizing
IGD+	✓	✓	✓	minimizing
Hypervolume	✓	✓		maximizing
HR	✓	✓		maximizing
ER	✓	✓		maximizing
RNI		✓		maximizing
R2	✓	✓		minimizing
Spread		✓		minimizing
UD		✓		maximizing

As mentioned in the introduction, MOEAs are used to solve MOPs. In the sequence, we detail the benchmarks and real-world problems that were used in the experiments of this work.

2.5 Multi-Objective Problems

There are several MOPs available in the literature, varying from logistics, engineering and mathematics. Here, some of them are introduced. They are here classified as *benchmarks* or real-world problems.

2.5.1 Benchmarks

Artificially constructed test problems offer many advantages over real-world problems for the purpose of general performance testing (BRADSTREET et al., 2007). With generic test suites, researchers can compare their multi-objective numerical and combinatorial optimization problem results (regarding effectiveness and efficiency) with others, over a spectrum of algorithms instances (COELLO, 2007).

Over the years, several benchmarks have been proposed. In (ZITZLER; DEB; THIELE, 2000), the ZDT suite was introduced; it is composed of some bi-objective continuous optimization problems that could be instantiated considering a different number of decision variables. However, this benchmark does not provide enough difficulty (HUBAND et al., 2006).

This was the motivation for the proposal of the DTLZ (DEB, K. et al, 2005) benchmark, which is composed of seven continuous optimization problems. In this suite, all the problems can be instantiated considering a different number of objectives and decision variables.

The CEC2009 (ZHANG et al., 2008), also named as UF, is a continuous optimization benchmark composed of ten bi-objective problems which can be instantiated using a different number of decision variables.

The Walking Fish Group (HUBAND et al., 2006) (WFG) benchmark is a flexible and scalable suite for continuous optimization, composed of nine problems with different Pareto optimal geometry, separability (if a Pareto front is disconnected or not) and modality. These problems can be instantiated using different numbers of objectives and decision variables.

Tables 3 and 4 presents respectively scalable and non-scalable benchmark functions in terms of objectives (no) and literature recommendations for the number of decision variables and other parameters. These tables also show the Pareto front shape for these benchmark functions.

2.5.2 Real-world problems

Over the years, several artificially constructed test problems have been proposed to compose benchmarks for evaluating meta-heuristics. These problems offer many advantages over real-world problems for the purpose of general performance testing (BRADSTREET et al., 2007), by allowing users to compare the results of their algorithms (regarding effectiveness

Table 3: Characteristics of benchmarks with scalable objectives

Problem	Variables (dv)	Properties
WFG1	$(no-1)*2+20$	convex, mixed
WFG2	$(no-1)*2+20$	convex, disconnected
WFG3	$(no-1)*2+20$	linear, degenerate
WFG4	$(no-1)*2+20$	concave
WFG5	$(no-1)*2+20$	concave
WFG6	$(no-1)*2+20$	concave
WFG7	$(no-1)*2+20$	concave
WFG8	$(no-1)*2+20$	concave
WFG9	$(no-1)*2+20$	concave
DTLZ1	$no+4$	linear
DTLZ2	$no+9$	non-convex
DTLZ3	$no+9$	non-convex
DTLZ4	$no+9$	non-convex
DTLZ5	$no+9$	degenerate
DTLZ6	$no+9$	disconnected
DTLZ7	$no+19$	disconnected

Table 4: Characteristics of benchmarks with fixed amount of objectives

Problem	Objectives (no)	Variables (dv)	Other parameters	Properties
ZDT1	2	30	-	convex
ZDT2	2	30	-	non-convex
ZDT3	2	30	-	convex, disconnected
ZDT4	2	10	-	non-convex
ZDT6	2	10	-	non-convex
UF1	2	30	-	concave
UF2	2	30	-	concave
UF3	2	30	-	concave
UF4	2	30	-	convex
UF5	2	30	n=10, ϵ=0.1	linear, disconnected
UF6	2	30	n=2, ϵ=0.1	linear, disconnected
UF7	2	30	-	linear
UF8	3	30	-	convex
UF9	3	30	ϵ=0.1	linear, disconnected
UF10	3	30		convex

and efficiency) with others, over a spectrum of algorithms instantiations (COELLO, 2007).

Even if benchmarks can test and compare how algorithms successfully solve their suite of problems, this does not guarantee these algorithms' effectiveness and efficiency in solving real-world problems (COELLO, 2007).

In the following, some real-world problems that were used in this work are presented.

2.5.2.1 moTSP

The traveling salesperson problem (TSP) is probably the most famous and extensively studied problem in the field of Combinatorial Optimization (GUTIN; PUNNEN, 2006). This problem consists of a set of cities to be visited by a salesperson. Here, from a starting city (e.g., the hometown), the salesperson has to visit all cities once, and then returning to the starting city. The challenge of the problem is to reduce the cost of visiting the cities by finding the best order to visit. Figure 13 presents an example of a TSP route departing from a starting point, visiting all cities and returning to the starting point.

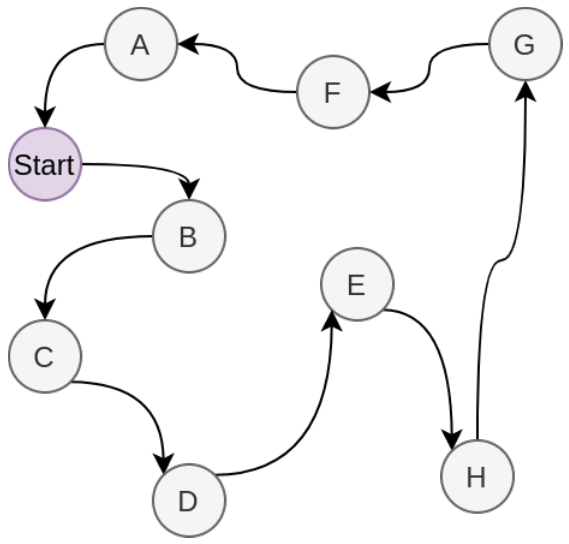


Figure 13: Traveling Salesperson Problem

Formally, the TSP problem is defined by a complete graph $G(V, E)$, where V is the set of ne elements composed by the nodes (or cities) to be visited and the departing node (depot or the starting city). A is the set of edges connecting the nodes. Finally, $c_{i,j}$ represents the cost necessary for going from a given node i to the node j. Thus, $\sum_{i=0}^{ne} c_{i,j}$ determines the tour cost, which is the objective function in this problem.

The Multi-objective Traveling Salesperson Problem (moTSP) (HANSEN, 2000) is a generalization of the TSP. Here, instead of using just one cost value, the problems can use, at the same time, several costs. Thus, costs as distances, the necessary time to deliver, or other relevant objectives make TSP a multi-objective problem.

The TSP can either be considered a real-world problem or a benchmark. The difference is how the problem is instantiated. For example, if latitude, longitude, and real-world distances between cities are considered, then we have a real-world problem. If artificial values are considered, then we have a benchmark.

2.5.2.2 Crashworthiness

The vehicle crashworthiness problem (LIAO et al., 2008) is a three-objective non-constrained problem where the crash safety level of a vehicle is optimized. In this problem, a higher safety level means how well a vehicle can protect the occupants from the effects of a frontal accident. In this problem, there are five decision variables (represented as t_i in Figure 14) that represent the thickness of reinforced members around the car front; and three objective functions to minimize: (i) the mass of the vehicle, (ii) integration of collision acceleration in the full-frontal crash, (iii) the toe-board intrusion in the 40% offset-frontal crash.

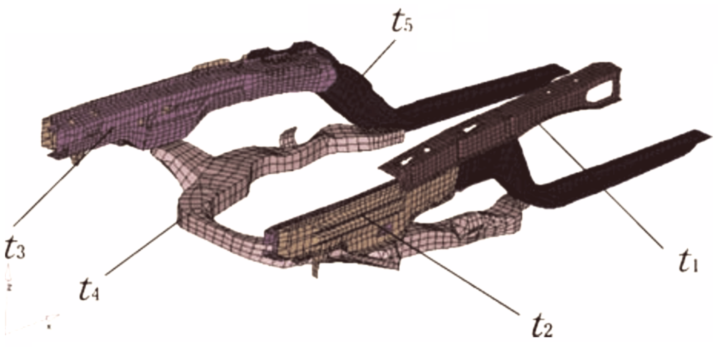


Figure 14: Vehicle crashworthiness problem, adapted from (LIAO et al., 2008)

2.5.2.3 Car Side Impact

The Car Side Impact Problem (JAIN; DEB, 2014) (Figure 15) is a three-objective constrained problem which involves minimizing the weight of the car, the pubic force experienced by a passenger and the *V-Pillar* average speed, responsible for withstanding the impact load. This problem is defined as eleven decision variables describing the thickness of several car parts, such as B-pillars (inner and reinforcement), floor side inner, cross-members, door (beam and reinforcement), roof rail, the material of B-Pillar and floor side inner, and variables representing the barrier: the height and hitting position.

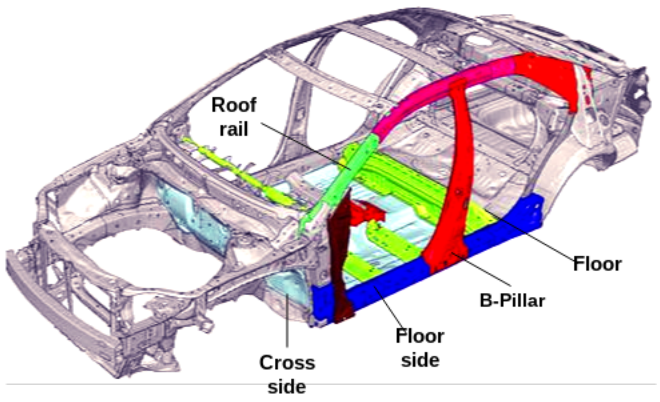


Figure 15: Car Side Impact, adapted from (JAIN; DEB, 2014)

2.5.2.4 Machining

The machining problem (GHIASSI et al., 1984) formulates machining recommendations under multiple criteria. This problem considered tests on the aluminum cut with *VC-3 carbide* cutting tools as a basis to test the approach, which has significant automotive industry applications (GHIASSI et al., 1984). This problem has three decision variables, where speed (v), feed (j) and depth of cut (d) are attributes considered in their definition. Four objectives are considered: (i) minimizing the surface roughness; (ii) maximizing the surface integrity, which refers to the amount of undamaged primary silicon at and immediately below the surface; (iii) maximizing the tool life, which is generally defined as the machining time to reach a fixed amount of uniform flank wear; and (iv) maximizing the metal removal rate, which is a measure of parts made per unit machining time.

2.5.2.5 Water

The Water Resource Planning (TAPABRATA; KANG; SEOW, 2001) (Figure 16) is a five-objective constrained problem which involves optimal planning for a storm drainage system in an urban area. The problem variables are the local detention storage capacity x_1, the maximum treatment rate x_2 and the maximum allowable overflow rate x_3. There are five objective functions to be minimized: (i) the drainage network cost, (ii) the storage facility cost, (iii) the treatment facility cost, (iv) the expected flood damage cost, and (v) the expected economic loss due to flood.

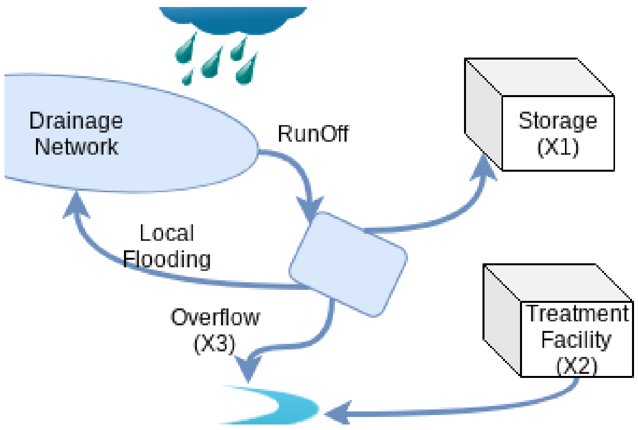


Figure 16: Water problem

2.5.2.6 Four Bar Truss

Four Bar Truss (STADLER; DAUER, 1993) is a continuous optimization problem with four decision variables. Figure 17 presents this problem composed of the following two objectives: (i) The volume of the truss (ii) the displacement Δ of the joint connecting bar 1 and 2 subject to given physical restraints on the feasible cross-sectional areas $\{x_1, x_2, x_3, x_4\}$ of the four bars. L and F are respectively the length of each bar and force of magnitude (here $F = 10kN$).

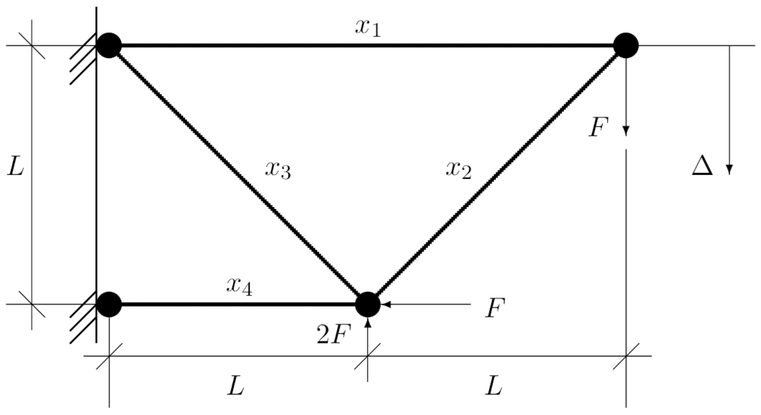


Figure 17: Four Bar Truss, extracted from (ENGAU, 2007)

2.5.2.7 Golinski

Golinski (GOLINSKI, 1970) is a gear reducer problem modeled as a bi-objective continuous optimization problem and composed of seven decision variables and eleven constraints. The speed reducer problem presents the design of a simple gearbox of a small aircraft engine, which allows the engine to rotate at its most efficient speed. This has been used as a testing problem for nonlinear optimization methods in the literature (GUNAWAN; FARHANG-MEHR;

AZARM, 2003).

2.5.2.8 Quagliarella

Quagliarella (QUAGLIARELLA; VICINI, 1998) is a wing design modeled as a bi-objective continuous optimization problem where the two objectives are related to aerodynamic and structural requirements. The resulting wing is then modified to further reduce its aerodynamic drag. This problem is composed of sixteen decision variables and has a Pareto shape defined as disconnected and convex.

2.5.2.9 Poloni

Poloni (POLONI; MOSETTI; CONTESSI, 1996) is the design of a multi-point airfoil modeled as a bi-objective continuous optimization problem where two objectives are considered: (i) to have a high lift at low speed (ii) having a low drag at transonic speed. This problem is composed of two decision variables and has a Pareto shape defined as disconnected and concave.

2.5.2.10 The Black Box Optimization Competition

The Black Box Optimization Competition (IFN, 2017) consists of ten challenging real-world bi-objective optimization problems from various domains. Table 5 presents these problems with their number of variables. All the problems are in continuous space.

Table 5: The Black Box Optimization problems

Problem name	Source
Vibrating Platform Design	(GUNAWAN; AZARM, 2005; IFN, 2017)
Optical Filter	(GIOTIS et al., 2001; IFN, 2017)
Welded Beam Design	(DEB; SUNDAR, 2006; IFN, 2017)
Disk Brake Design	(YANG, 2013; IFN, 2017)
Heat Exchanger	(IFN, 2017)
Hydro Dynamics	(IFN, 2017)
Area Under Curve	(IFN, 2017)
Kernel Ridge Regression	(IFN, 2017)
Facility Placement	(IFN, 2017)
Neural Network Controller	(IFN, 2017)

2.6 Discussion

Table 6 summarizes all the mentioned real-world problems by describing the number of objectives, variables and constraints. These problems were selected due to the diversity of areas contemplated. For example, *CarSideImpact*, *CrashWorthiness*, and *Disk Brake Design* are problems from mechanical engineering. *Water* and *Hydro Dynamics* belong to Naval Engineering. *Facility Placement* and *moTS*P come from logistics. *Kernel ridge regression* and *Neural Network controller* are from Computer Science. Thus, using several problems from different areas give us a better way to place newly proposed algorithms in a realistic scenario. It also makes these algorithms more attractive to researchers from these areas since they usually aim to solve their problems rather than test new algorithms.

Table 6: A brief description real-world multi-objective problems containing the number of objectives, variables and constraints

Problem name	Objs.	Variables.	Constraints
moTSP	Scalable	Scalable	Scalable
Water	5	3	7
Machining	4	3	3
CarSideImpact	3	7	0
CrashWorthiness	3	5	0
FourBarTruss	2	4	0
Golinski	2	7	11
Quagliarella	2	16	0
Poloni	2	2	0
Vibrating Platform Design	2	5	0
Optical Filter	2	11	0
Welded Beam Design	2	4	0
Disk Brake Design	2	4	0
Heat Exchanger	2	16	0
Hydro Dynamics	2	6	0
Area Under Curve	2	10	0
Kernel Ridge Regression	2	5	0
Facility Placement	2	20	0
Neural Network Controller	2	24	0

2.7 Conclusion

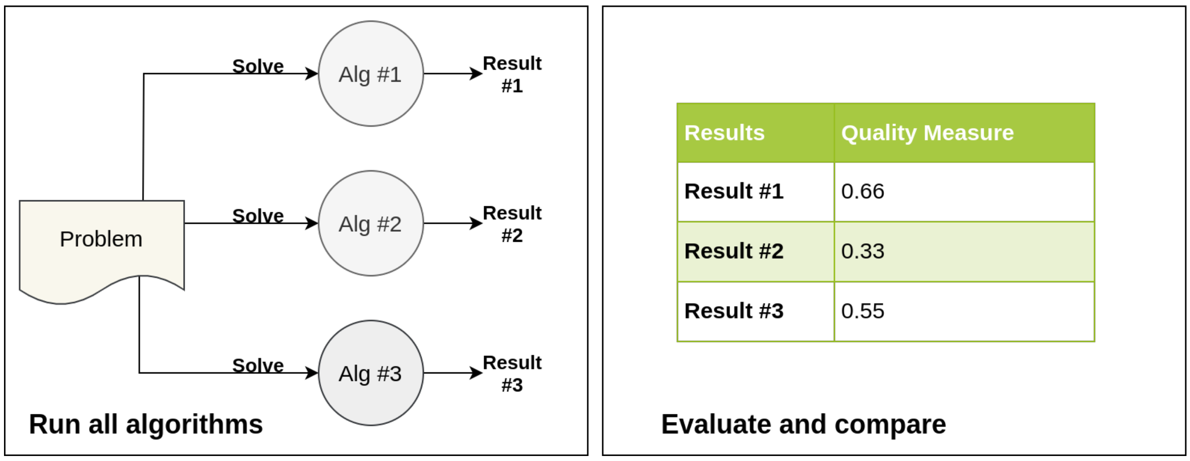
This chapter presented several concepts related to Evolutionary Computation, such as optimization, multi evolutionary algorithms, quality indicators and multi-objective problems. All these subjects are necessary to understand this work.

The next chapter introduces the algorithm selection problem and a way to tackle this problem: *hyper-heuristics*, the category in which this present work belongs.

3 ALGORITHM SELECTION PROBLEM

It has long been recognized that there is no single optimization algorithm that can achieve the best performance in all problem instances, and algorithms perform differently depending on the problem characteristics (WOLPERT; MACREADY, 1997). However, it is possible to achieve better results, on average, across many different classes of a problem, when we tailor the selection of an algorithm to the characteristics of the problem instance (SMITH-MILES et al., 2009).

Moreover, selecting an algorithm to solve an optimization problem instance is not a trivial task. Usually, a tuning method is necessary, if there is no previous knowledge about which algorithm to use and what is the recommended algorithm configuration to solve a given optimization problem. The tuning task (Figure 18) consists of solving an optimization problem using different algorithm instances, taking their results, and finding the best according to a quality measure.



Results	Quality Measure
Result #1	0.66
Result #2	0.33
Result #3	0.55

Figure 18: Tuning Task

There are also motivations from the *No Free Lunch Theorem* which establish that "for any algorithm, any elevated performance over one class of problems is an offset by diminished performance over another class" (WOLPERT; MACREADY, 1997).

3.1 Definition

Rice (RICE, 1976) first posed the question: *" Which algorithm is likely to perform best for my problem?"*. From this point, he designed *The Algorithm Selection Problem* (ASP). The basic definition for ASP is defined as $ASP = \langle \mathcal{P}, \mathcal{A}, \mathcal{E} \rangle$ which is composed by the following elements:

- Problem Space ($\mathcal{P}$): A vast and diverse set of problems with all possible problem instances;
- Algorithm Space ($\mathcal{A}$): The vast and diverse set of all possible algorithms which can be used to solve a problem instance;
- Performance Evaluator ($\mathcal{E}$): The quality indicator set used to measure the performance of a particular algorithm for a given problem instance.

Figure 19 shows the ASP. In this figure, a selection $\mathcal{S}(p)$ has to be made for the problem instance $p \in \mathcal{P}$ considering the algorithm space $\mathcal{A}$. The algorithm selection considers performance measures in $\mathcal{E}$ to get the algorithm performance for each $a \in \mathcal{A}$ by mapping $e(a,p)$ and finally getting the value of performance $r = ||e(a,p)||$. $\mathcal{R}$ is composed by all values of performance r.

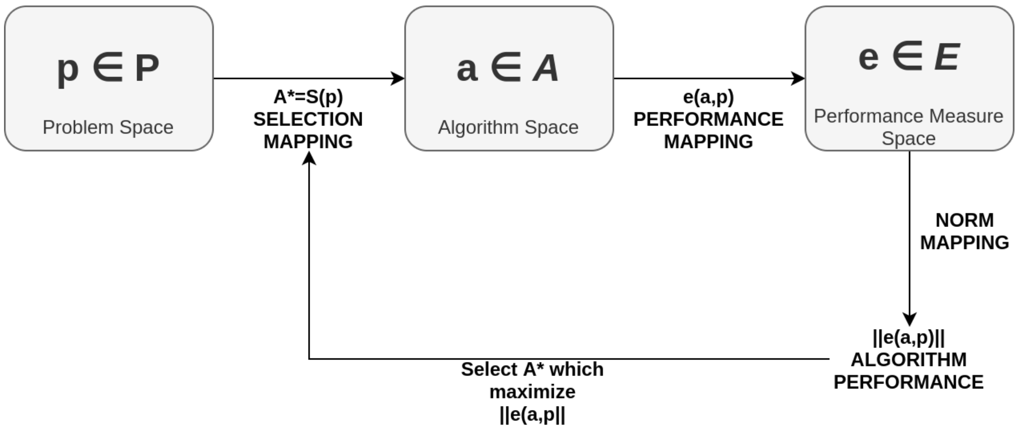


Figure 19: Rice's ASP basic model, adapted from (RICE, 1976)

The best algorithm selection, for example the tuning task mentioned before (Figure 18), it can be achieved by using the equation $(a_i \in \mathcal{A} \mid \forall a_j \in \mathcal{A} \mid ||e(a_i,p)|| \geq ||e(a_j,p)||)$, for a given problem instance $p \in \mathcal{P}$.

In (RICE, 1976), the ASP was also extended to contain the feature space ($\mathcal{F}$), necessary for some categories of problems (e.g., Classification/Regression problems). This feature space contains measurable characteristics of the instances generated by a computational feature extraction process applied to P. It may be viewed as a way to systematize the problem in a basic model. In this extended ASP, there is a set of features $f(p) \in \mathcal{F}$ and the selection mapping is given by $\mathcal{S}(f(p))$. Finally, in this context the ASP is defined as $\langle \mathcal{P}, \mathcal{F}, \mathcal{A}, \mathcal{E} \rangle$.

The ASP has been extended over the years by different Artificial Intelligence areas. Among them, we can highlight meta-learning and hyper-heuristics approaches (SMITH-MILES, 2012).

3.2 Meta-learning

In the machine learning community, the research in the field of meta-learning (learning about learning) differs from the traditional base-learning approach in their adaptation level. While learning at the base level is focused on accumulating experience on a specific learning task, learning at the meta-level accumulates experience in the performance of multiple applications of a learning system (BRAZDIL et al., 2008). This area has mostly focused on classification problems (SMITH-MILES, 2012).

Consider the following steps to solve a classification or regression problem. First, some data have to be prepared, by cleaning and formatting procedures, and an appropriate model has to be defined. This model is typically trained by solving a core optimization problem that optimizes the decision variables set of the model with respect to the selected loss function, and possibly some regularization function (BENNETT; PARRADO-HERNÁNDEZ, 2006).

Overall, the main steps of classification/regression are: (i) build a model hypothesis, (ii) define the objective function, (iii) solve the optimization problem by finding a maximum/minimum for the objective function in order to determine the parameters of the model. The first two steps are related to modeling problems of machine learning, while the third step is to solve the desired model by optimization methods (SUN et al., 2019). Thus, optimization problems lie at the heart of most machine learning approaches (BENNETT; PARRADO-HERNÁNDEZ, 2006). To solve the inner optimization problem, researchers have been using techniques such as Bayesian Optimization (MOČKUS, 1975), Evolutionary Algorithms (GOLDBERG; HOLLAND, 1988), and Gradient descent (CAUCHY, 1847), one of the most popular algorithms in this context (RUDER, 2016).

A meta-learning can be applied, for example, in classification ($\mathscr{P}$), solved using typical machine learning classifiers ($\mathscr{A}$), such as decision trees, neural networks, or support vector machines, where supervised learning methods have been used to learn the relationship between the statistical and information-theoretic measures of the instances ($\mathscr{F}$) and the accuracy ($\mathscr{E}$) of the classifier algorithms (SMITH-MILES, 2012). Hence, algorithm selection/recommendation helps the user to choose the learner by generating a ranking of algorithms or indicating a single algorithm according to their predictive performance (PAPPA et al., 2014).

3.3 Hyper-heuristics

Hyper-heuristics are defined as a high-level methodology that, given a particular problem instance and some low-level heuristics (LLH), automatically produces an adequate combination of them to solve the problem efficiently (BURKE et al., 2013; DRAKE et al., 2019).

Hyper-heuristics can learn how to combine LLHs by using feedback from the search process. According to the source of the feedback, this algorithm can be classified as online or offline. There is also hyper-heuristics which do not perform any kind of learning. Hyper-heuristics can also be classified according to its strategy: selection or generation-based, or by the kind of employed heuristics: construction or perturbation (BURKE et al., 2013; DRAKE et al., 2019). Figure 20 presents the hyper-heuristic classification.

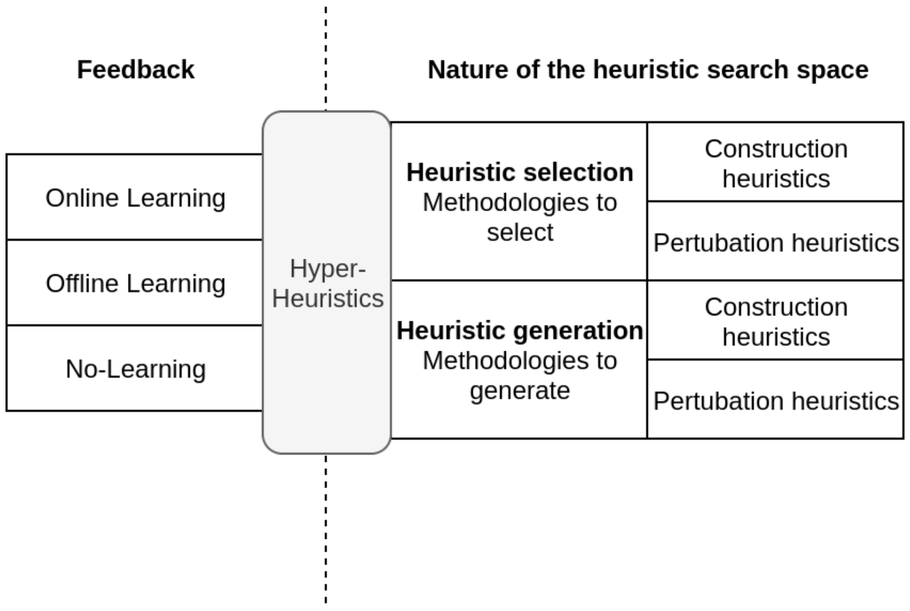


Figure 20: Hyper-heuristic classification, adapted from (BURKE et al., 2013)

In an online approach, the learning takes place while the algorithm is solving an instance of a problem. In offline learning, similarly to supervised learning, the knowledge is gathered considering a set of training instances that hopefully generalize to solving unseen instances (BURKE et al., 2013). No learning hyper-heuristics use a fixed rule to select or generate low-level heuristics (BURKE et al., 2010).

Another characteristic of hyper-heuristics is the heuristic type to be selected/generated. Constructive heuristics build solutions by selecting a component at each search step. Perturbative heuristics transform a complete solution into another complete solution, i.e., constructive heuristics accept and generate partial solutions, while perturbative heuristics allow only complete solutions (BURKE et al., 2010).

One important concept to consider when designing hyper-heuristics is the domain-barrier (BURKE et al., 2010). The domain-barrier concept, shown in Figure 21, guarantees that the hyper-heuristic does not have any information about the optimization problem, such as objective functions and solutions. Thus, hyper-heuristics only has knowledge about the number of heuristics, their performance values and other problem independent data (COWLING; KENDALL; HAN, 2002). Problem information can only be accessed by LLHs. That is necessary because using problem knowledge makes it very difficult to generalize or apply meta-heuristics over different/new problems, causing meta-heuristics to be re-developed in order to solve problems from a different domain (SABAR et al., 2014).

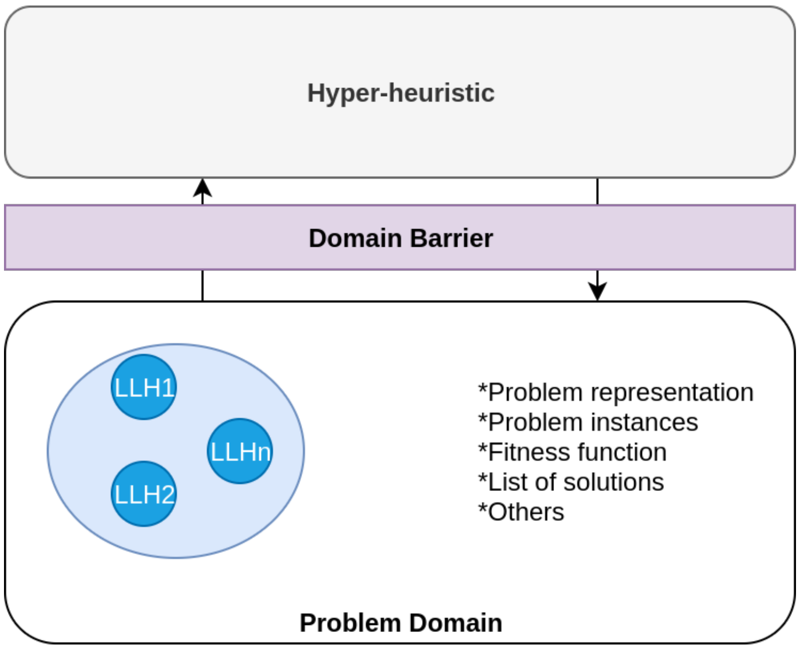


Figure 21: Domain barrier, adapted from (NOTTINGHAM, 2011)

Online Selection hyper-heuristics are the most popular hyper-heuristic subgroup (DRAKE et al., 2019), and it is the category where this thesis belongs. Algorithm 1 presents how a generic online selection HH works. This algorithm has as input: an optimization problem p, the $\mathcal{LLH}$ set, a criterion $\mathcal{S}c$ used to define how much a $llh \in \mathcal{LLH}$ can execute, and a set of evaluators $\mathcal{E}$ used to determine how well $\mathcal{LLH}$ performed.

First, the algorithm starts by generating random solutions for the given problem p (Line 3) and initializing the set of evaluations $\mathcal{R}$ generated by a member of $\mathcal{E}$. Then, while a stopping criterion is not satisfied, the criteria $\mathcal{S}c$ defines strategies for $llh \in \mathcal{LLH}$ considering the current set of evaluations $\mathcal{R}$ (Line 6), elements from $\mathcal{LLH}$ generate new offspring (Line 7), and evaluators in $\mathcal{E}$ evaluate the $\mathcal{LLH}$ outcome and store their evaluations in $\mathcal{R}$ (Line 8).

Algorithm 7: Generic Online Selection Hyper-Heuristic

```
1  Input: $p, \mathcal{LLH}, \mathcal{S}c, \mathcal{E}$
2  begin
3      Initialize the population with random solutions to solve problem $p$;
4      Initialize $\mathcal{R}$;
5      while a termination condition is not satisfied do
6          $\mathcal{S}c$ define strategies for $llh \in \mathcal{LLH}$ considering $\mathcal{R}$;
7          Use one or more $llh \in \mathcal{LLH}$ to generate new solutions;
8          Evaluate solutions according to evaluators in $\mathcal{E}$ to generate $\mathcal{R}$;
9      end
10 end
```

Most of the works treat heuristics, such as differential evolution, crossover and mutation as low-level heuristics. However, some works treat algorithms as low-level heuristics (DRAKE et al., 2019). However, by viewing heuristics as algorithms, some authors have occasionally tried to argue for the absolute superiority of one heuristic over another (BURKE et al., 2003). In this case, the term 'heuristic' is sometimes used to refer to a whole algorithm and it is sometimes used to refer to a particular decision process sitting within some repetitive control structure (BURKE et al., 2003). This present work belongs to this category.

If we considerate online selection hyper-heuristics for choosing algorithms, there are two possibilities for dealing with Evolutionary Algorithms (as LLHs): (i) the $\mathcal{S}c$ defines one $llh \in \mathcal{LLH}$ to run by turn, and before running, it receives the full main population of solutions, generates new solutions and update this population, and then return the new population to the HH. Chapter 6 presents some HHs of this category; (ii) the $\mathcal{S}c$ defines a set of LLHs to act on the same population at the same time. For this purpose, it splits the main population into subpopulations, sends each subpopulation to one llh. So, each llh generates new solutions, updates the subpopulation, and returns it to the HH. This thesis belongs to this last category.

3.4 Meta-learning vs Hyper-heuristics

Meta-learning and hyper-heuristics are, in fact, similar because both tackle the ASP. However, they differ in how they deal with an *optimization problem*. Hyper-heuristics are focused on selecting heuristics or meta-heuristics aiming to solve optimization problems, while meta-learning is more concerned about selecting or combining algorithms to solve tasks such as classification, the most studied task in meta-learning (PAPPA et al., 2014).

Aiming to solve optimization problems, a hyper-heuristic can be designed to select heuristics to solve inner optimization existing in classification/regression problems. Hyper-heuristics, however, has not been designed for selecting/generating machine learning models, the task well performed by meta-learning approaches. For example, two well-known algorithms that follow this approach are bagging and boosting (PAPPA et al., 2014).

In terms of meta-learning, some research employs voting methods in order to solve classification problems. For example, in (DIMITRIADOU; WEINGESSEL; HORNIK, 2002), the authors proposed a voting scheme for cluster algorithms allowing several runs of cluster algorithms resulting in a shared partition and aiming to tackle the problem of choosing an appropriate clustering method. In (SEVILLANO; ALÍAS; SOCORÓ, 2007), the authors designed a soft cluster ensemble that ranks the instances according to their membership probability concerning each cluster, and conducted thorough experimental comparisons on voting-based, in this case, the Borda voting scheme. In (DOMÍNGUEZ; CARRIÉ; PUJOL, 2009), the author also considered Condorcet methods and got similar results as obtained by Borda. These mentioned works also tackle the algorithm selection problem using voting procedures, which is similar to what this thesis is concerned about. However, differently from the mentioned work, this thesis aims to propose a hyper-heuristic, and by doing that, focus on optimization problems, more especially, multi-objective optimization.

3.5 Conclusion

This chapter introduced the Algorithm Selection Problem concepts, one of the most important theoretical backgrounds for meta-learning and hyper-heuristic. Hyper-heuristics were also defined and a generic model for online selection hyper-heuristics was defined.

The next chapter presents multi-agent and voting concepts. This comprehension is also necessary for the understanding of further chapters.

4 MULTI-AGENT SYSTEMS AND SOCIAL CHOICE THEORY

This present chapter introduces the concept of multi-agent systems (Section 4.1) and Artifacts, necessary to understand the proposed work. This chapter also introduces Social choice theory concepts (Section 4.2), in special the voting procedures Borda, Copeland and Kemeny, employed by the proposed hyper-heuristic (Chapter 5).

4.1 Multi-Agent Systems

An agent is a computer system that is situated in some environment, and that is capable of autonomous action in this environment in order to meet its delegated objectives (WOOLDRIDGE, 2009). An agent perceives the environment by sensors in order to understand it. Thus, based on his perceptions and considering preconditions, the agent can plan how to act, that means, use or not on a given actuator in order to update the environment. Figure 22 show the agent-environment interaction.

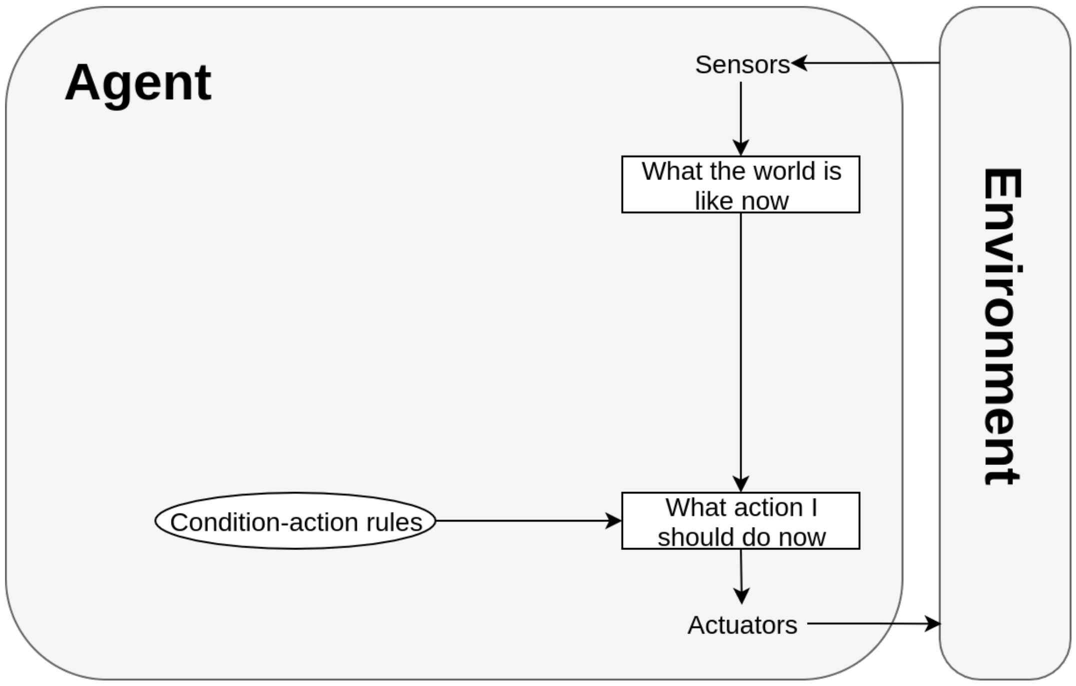


Figure 22: Agent-environment interaction, adapted from (RUSSELL; NORVIG, 2010)

A multi-agent system (MAS) is one that consists of a number of agents, which interact with one another, typically by exchanging messages through some computer network infrastructure (WOOLDRIDGE, 2009). According to (DEMAZEAU, 1995), a multi-agent system can be developed considering four distinct dimensions, making a MAS a composition of Agents + Environment + Interactions + Organization, or $A + E + I + O$, the Vowel methodology. The Vowel methodology elements have the following characteristics:

- The agent models, or agent architectures, range from simple fine-grained automaton to complex coarse-grained knowledge-based systems (DEMAZEAU; MULLER, 1990);
- The environments are domain dependent (DEMAZEAU, 1995);
- Interaction structures and languages ranging from physics-based interactions to speech acts (DEMAZEAU, 1995);
- Organizations range from dynamic ones inspired by biological studies, to more governed by social laws ones inspired by sociological studies (DEMAZEAU, 1995).

Thus, a multi-agent system will be composed of several agents, an environment, a set of possible interactions, and possibly at least one organization (DEMAZEAU, 1995).

The task of building multi-agent systems often makes necessary the specification about how agents percept the environment, communicate with one another, and update the environment. Some interesting approaches can be found in the literature and, among them, the concept of an *Artifact*.

Artifacts are non-autonomous, function-oriented, stateful entities, that represent the environment elements. They are controllable, observable, and used to model the tools and resources used by agents (RICCI, A. and VIROLI, M. and OMICINI, A., 2007). Thus, artifacts can be used to design and program a suitable agent working context or environment: a set of objects (in the wide sense) and tools that agents can share and use to support their individual as well as social activities (RICCI; VIROLI; OMICINI, 2006).

Due to their characteristics, artifacts can be employed to provide communication, such as messages and blackboards, and providing tools for coordination, such as auctions or elections.

4.2 Social Choice Theory

Social Choice theory is the study of systems and institutions for making collective choices (KELLY, 2013). This theory is responsible for studying voting systems and their applications.

A voting system is a very common way of resolving disagreements, determining common opinions, choosing public policies, electing office-holders, finding winners in contests, and solving other problems of amalgamating a set of (typically individual) opinions (NURMI, 2010).

The simplest way to perform an election is majority voting. However, in an election, every candidate can be beaten by others based on sincere simple majority voting. In order to solve this problem, different voting methods were proposed. In the following, some of them are presented.

4.2.1 Borda Count method

The Borda Count (BORDA, 1784) is a voting method that uses a ranking of preferences, which means, electors rank their candidates preferences. Thus, for each ordering, the best candidate receives $n-1$, where n is the number of candidates, the second candidate receives $n-2$, the third $n-3$, and so on. The chosen candidate is the one with the highest sum of preference points. Equation 4.1 shows how each candidate is ranked, where m is the number of electors, n the number of candidates, and $r_{i,j}$ is the rank position of a given candidate i according to an elector m from the electors group M.

$$rank(i) = \sum_{i=1}^{m} n - r_{i,m} \tag{4.1}$$

After we have Borda scores for all candidates, they are sorted from higher to lower in order to generate the final ranking. The selected candidate is the top-ranked candidate, as shown in Equation 4.2.

$$winner(i) = \forall j \in N rank(i) > rank(j) \tag{4.2}$$

4.2.2 Condorcet methods

The Condorcet's principle says that if a candidate defeats every other candidate in pairwise comparisons (a Condorcet winner), it must be elected (MOULIN, 1988). Thus, after every voter vote, a pairwise comparison is performed. In the following example, we have three different candidates and voters. Then we perform the one-on-one contest for each candidate in all voters.

- Candidate 1 vs Candidate 2 according to Voter 1;
- Candidate 1 vs Candidate 2 according to Voter 2;
- Candidate 1 vs Candidate 2 according to Voter 3;
- Candidate 1 vs Candidate 3 according to Voter 1;
- Candidate 1 vs Candidate 3 according to Voter 2;
- Candidate 1 vs Candidate 3 according to Voter 3;
- Candidate 2 vs Candidate 3 according to Voter 1;
- Candidate 2 vs Candidate 3 according to Voter 2;
- Candidate 2 vs Candidate 3 according to Voter 3;

Thus, the Condorcet's principle has as the fundamental idea that the opinion of the majority should prevail, at least when majority comparisons pinpoint an unambiguous winner (MOULIN, 1988).

4.2.2.1 Copeland method

The Copeland Voting method (COPELAND, 1951) follows the Condorcet's principle. In order to follow this principle, Copeland's method ranks the alternatives according to their score in the sum of rows in the antisymmetric matrix of the Condorcet relation (POMEROL; BARBA-ROMERO, 2012).

To perform a Copeland voting, first we create the Condorcet's pairwise comparison among candidates. For example, considering two given candidates c_i and c_j, the pairwise is calculated according to:

$$S(i,j) = \left\{ \begin{array}{l} 1 \text{ if } c_i \text{ is better than } c_j \\ -1 \text{ se } c_i \text{ is worse than } c_j \\ 0 \text{ otherwise} \end{array} \right\} \tag{4.3}$$

After calculating all pairwise comparisons, for all candidates (c_i), we find the candidate Copeland score according to:

$$CS(i) = \sum_{i \neq k} S(i,k) \tag{4.4}$$

Posteriorly, we have Copeland scores for all candidates. Then, the Copeland scores are sorted from higher to lower in order to generate the final ranking. The selected candidate is the top-ranked candidate, as shown in Equation 4.5.

$$winner(i) = \forall j \in NCS(i) > CS(j) \tag{4.5}$$

4.2.2.2 Kemeny-Young method

The Kemeny-Young voting procedure (KEMENY, 1959) also follows the Condorcet's principle. So, this procedure also starts the Condorcet pairwise comparison, generating for each elector a ranking of preferences. For example, given four candidates a, b, c and d; an elector e_1 can define his preferences as $b \succ a \succ d \succ c$, other elector ($e_2$) can define his preference as $c \succ a \succ d \succ b$, a third elector ($e_3$) defines his preferences as $c \succ a \succ b \succ d$, and so on.

In the following, this voting procedure performs the sum of all pairwise preferences. For example, all the three electors prefer a instead of d ($a \succ d$), two electors prefer $c \succ \{a, b, d\}$, all these comparisons will generate a sum of preferences. All possible pairwise comparison is performed even if no elector preferred it. Table 7 shows these sums.

Table 7: Candidates comparison scores

Candidate	a	b	c	d
a	-	2	1	3
b	1	-	1	2
c	2	2	-	2
d	0	1	1	-

Finally, the overall sum is created (Table 8). This overall sum is created for every possible combination of preferences, making this ranking creation a combinatorial problem. To calculate an overall sum for a given preference rank, we sum all comparison scores existing in the given preference rank. For example, to calculate the overall value for the preference ranking $c \succ a \succ b \succ d$, we need to sum the comparison scores (from Table 7) for $c \succ \{a, b, d\}$, resulting an overall value of 6 $(2 + 2 + 2)$, plus $a \succ \{b, d\}$, resulting 5 $(2 + 3)$, plus $b \succ d$, resulting 2. So, the rank $c \succ a \succ b \succ d$ has an overall value of 13 $(6 + 5 + 2)$. The candidate elected is the one with the biggest overall value, in our case, $c \succ a \succ b \succ d$.

Table 8: Overall comparison

Ranking	Overall	Ranking	Overall	Ranking	Overall
$a \succ b \succ c \succ d$	11	$b \succ c \succ a \succ d$	11	$c \succ d \succ a \succ b$	9
$a \succ b \succ d \succ c$	10	$b \succ c \succ d \succ a$	8	$c \succ d \succ b \succ a$	8
$a \succ c \succ b \succ d$	12	$b \succ d \succ a \succ c$	6	$d \succ a \succ b \succ c$	6
$a \succ c \succ d \succ b$	11	$b \succ d \succ c \succ a$	7	$d \succ a \succ c \succ b$	7
$a \succ d \succ b \succ c$	9	$c \succ a \succ b \succ d$	13	$d \succ b \succ a \succ c$	5
$a \succ d \succ c \succ b$	10	$c \succ a \succ d \succ b$	12	$d \succ b \succ c \succ a$	6
$b \succ a \succ c \succ d$	10	$c \succ b \succ a \succ d$	12	$d \succ c \succ a \succ b$	8
$b \succ a \succ d \succ c$	9	$c \succ b \succ d \succ a$	9	$d \succ c \succ b \succ a$	7

4.3 Conclusion

This chapter introduced concepts of multi-agent systems, Artifacts, Social choice theory, and voting methods (Borda, Copeland and Kemeny).

In this thesis, multi-agent theory was considered in order to propose a distributed system, where components compete and cooperate for computational resources. In order to make the MAS design easier, Artifacts were employed to represent shared information. Since agents are also competing, Voting methods were employed in order to summarize preferences and solve disagreements.

Thus, all concepts presented here and on Chapters 2 and 3 are necessary to understand this thesis proposal: Multi-Agent Election Based Hyper-Heuristic.

PART II

PROPOSAL

5 MULTI-AGENT ELECTION BASED HYPER-HEURISTIC

This chapter details the Multi-Objective Agent-Based Hyper-Heuristic (MOABHH), a hyper-heuristic designed as a multi-agent system which employs voting concepts (Chapter 4) in order to tackle the algorithm selection Problem (Chapter 3) for multi-objective optimization (Chapter 2). In this approach, MOEAs (Section 2.3) are considered as LLH and multi-objective quality indicators (Section 2.4) as voters. In the following, this approach is formalized.

A preliminary description of the model was originally published in (CARVALHO; SICHMAN, 2017).

5.1 General Model

Formally, in order to solve a given problem $p \in \mathcal{P}$, MOABHH considers a set of MOEAs ($\mathcal{LLH}$) and a set of quality indicators voters ($\mathcal{V}$) as agents. Each $v_j \in \mathcal{V}$ assigns a voting preference $vp_{i,j} \in \mathcal{VP}$ for each $llh_i \in \mathcal{LLH}$. For each llh_i, this preference is based on the current population $rp \in \mathcal{RP}$ generated by the different v_j to solve p.

Considering $\mathcal{VP}$ and a predefined voting function $\mathfrak{vote}$, MOABHH generates an election outcome $\mathcal{EO}$ composed by all candidates sorted according to the election outcome from the winner to losers.

With the election outcome $\mathcal{EO}$, a hyper-heuristic agent can dynamically assign more or less participation percentage $pp \in \mathcal{PP}$ on generating new solutions for a given low-level heuristic $llh \in \mathcal{LLH}$. Thus, this model can be instantiated using different voting methods, quality indicators and low-level heuristics.

First, let's extend the ASP model $\langle \mathcal{P}, \mathcal{A}, \mathcal{E} \rangle$ and define the hyper-heuristic approach as the following:

$$\mathit{MOABHH} = \langle \mathcal{P}, \mathcal{LLH}, \mathcal{RP}, \mathcal{V}, \mathcal{VP}, \mathcal{EO}, \mathcal{PP}, \mathfrak{active}, \mathfrak{vote}, \mathfrak{election}, \mathfrak{part} \rangle,$$

where

- $\mathcal{P}$ is the optimization **problem** set to be solved;

- $\mathcal{LLH} = \{llh_i : i \leq n, n = |\mathcal{LLH}|\}$ is the set of **low-level heuristics** that generates new solutions[1]. At each time t, $\mathcal{LLH}^t_{active}$ represents the set of active low-level heuristics that generate new solutions at a given time t. This is defined according to the function $\mathfrak{active}$;

- $\mathfrak{active} : \mathcal{LLH} \times \mathcal{PP}^t \implies \mathcal{LLH}^t_{active}$ is a function that defines which $llh \in \mathcal{LLH}$ is **active** at a given time t[2].

- $\mathcal{RP}^t = \{rp_k : k \leq n\}$ is the set of **resulting populations** generated by each $llh \in \mathcal{LLH}^t_{active}$ at a given time t. For $llh \notin \mathcal{LLH}^t_{active}$, $rp^t_i = \emptyset$;

- $\mathcal{V} = \{v_j : j \leq m, m = |\mathcal{V}|\}$ is the set of **voters** responsible to evaluate LLHs[3]. A $\mathcal{V}$ member can, for example, be Hypervolume, Epsilon, or IGD quality indicators for multi-objective optimization, as described in section 2.4;

- $\mathfrak{vote} : \mathcal{V} \times \mathcal{RP}^t \implies \mathcal{VP}^t$ is a **voting** function that, for each voter $v \in \mathcal{V}$, generates its voting preferences for each $llh \in \mathcal{LLH}^t_{active}$ at a given time t. In the case where $rp^t_k = \emptyset$ we have $vp^t_{i,j} = 0$;

- $\mathcal{VP}^t = \{vp^t_{i,j} : i \leq m, j \leq n\}$ is the set of **voting preferences** generated by $\mathfrak{vote}$[4];

- $\mathcal{VM} = \{Borda, Copeland, Kemeny\}$ is the set of **voting methods**;

- $\mathfrak{election} : \mathcal{VM} \times \mathcal{VP}^t \implies \mathcal{EO}^t$ is the **election** function that generates an election outcome $\mathcal{EO}^t$ at a given time t, by the aggregation of the set of voting preferences $\mathcal{VP}^t$, given a certain voting method $vm \in \mathcal{VM}$;

- $\mathcal{EO}^t = \{eo^t_l : l \leq n\}$ is the **election outcome** generated by the function $\mathfrak{election}$. Each $eo^t \in \mathcal{EO}^t$ is defined as a tuple $eo^t_l =< llh, rank >$, where $rank \in [1...n]$;

- $\mathfrak{part} : \mathcal{EO}^t \times \mathcal{PP}^t \implies \mathcal{PP}^{t+1}$ is a function responsible for defining each $llh \in LLH$ its **participation** on generating new solutions in the next iteration;

- $\mathcal{PP} = \{pp^t_i : i \leq n, pp^t_i \in [0 \dots 100]\}$ is the set of $\mathcal{LLH}$'s **participation percentage** on generating new solutions.

[1]$\mathcal{LLH}$ corresponds to $\mathcal{A}$ on ASP model, as described in section 3.1.
[2]At time 0, all llh_i are active.
[3]$\mathcal{V}$ corresponds to $\mathcal{E}$ on ASP model, as described in section 3.1.
[4]$\mathcal{VP}$ corresponds to $\mathcal{R}$ on ASP model, as described in section 3.1.

5.2 General Architecture

MOABHH is an agent-based hyper-heuristic. So, Artifacts and Agents (Chapter 4) are used in the architecture. Figure 23 shows MOABHH's agents interaction, where there are three kinds of agents, who share four kinds of *artifacts*. In this figure, Agents are represented by circles and Artifacts by parallelograms. Solid arrows mean writing permission and dotted lines mean reading permission.

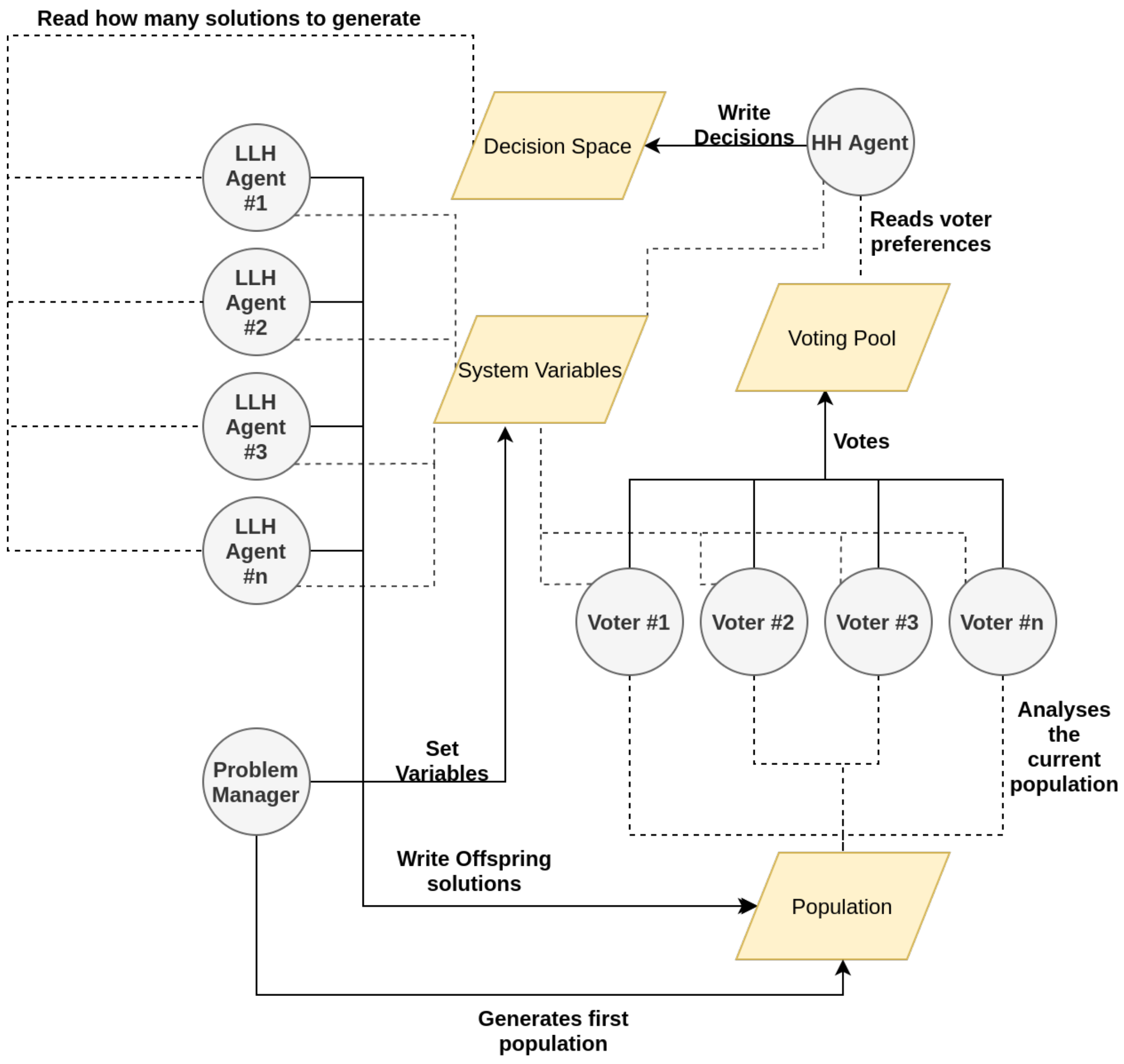


Figure 23: MOABHH General Architecture.

5.2.1 Artifacts

The four kinds of *artifacts* present in the architecture are the following:

- The *Population artifact* keeps the main current population of solutions (and sub-populations). This artifact is used by *Indicator voter agents*. When MOABHH starts, the Problem Manager agent randomly generates the first population and then assigns it to this artifact.
- The *Voting pool artifact* keeps all the voting preferences necessary to perform an election. This artifact is used by voter agents and the HH (Hyper-Heuristic) Agent, who defines which low-level heuristic win the election according to its voting rules.
- The *Decision space artifact* keeps all HH agent's decisions. All decisions can be read by a low-level heuristic agent and it defines how they have to proceed.
- The *System variables artifact* keeps the optimization problem; references points for quality indicators; MOABHH variables such as population size, the number of generations before voting (δ). This artifact is readable by all agents, and writable by the Problem Manager Agent.

5.2.2 Agents

The three kinds of *agents* present in the architecture are the following:

- The *Problem Manager agent* is responsible for receiving all MOABHH parameters and set them in the *System Variables* artifact.
- The LLH (*Low-Level Heuristic*) agent contains a particular meta-heuristic instance. This agent is responsible for generating a given number of solutions in a generation, where the number is defined by the HH agent. After generating new solutions, this agent adds the generated solutions to *Population Artifact*. All generated solutions are associated with their respective generator agent. Thus, it is possible to evaluate low-level heuristic performance.
- The *Voter* agent contains an instance of a quality indicator responsible for evaluating the LLHs outcome. Thus, all low-level heuristic agents have an associated quality value

that can be used for voting. To vote, this agent sends a ranking of its preferences following the quality values.

- The *Hyper-heuristic* agent uses information available on *Voting Pool Artifact* to employ a voting method. With the electoral outcome, this agent defines how many solutions each low-level heuristic can generate, giving to the best low-level heuristic more solutions to generate.

5.3 Processing phases

MOABHH has three processing phases: (i) Participation control, (ii) Evaluation and voting, (iii) Population allocation. These steps are detailed next.

5.3.1 Participation control

The $\mathcal{PP}$ set contains values from 0 to 100 representing the percent on generating solutions each LLH has in the next generations. In the beginning ($\mathcal{PP}^0$), all items from this set receive the same value. For example, if we have five LLHs ($n = 5$) then each $pp_i^0 \in \mathcal{PP}^0$ will receive 20.

The participation is changed along with the search, following election outcomes and increasing the participation for winners and decreasing it for losers. Figure 24 shows an example of this procedure. In the first step, all low-level heuristic receives the same number of solutions to generate. In the second and third steps, we can see that the best low-level heuristic (the blue one) can generate more solutions than others. In the fourth step, the best low-level heuristic was found and it is the only one executing.

The increasing/decreasing process considers the election outcome $\mathcal{EO}^t$, more precisely the position that a given LLH has got in the election. First, the last candidate in the election (n) has its decreasing calculation according to $pp_n^t * \beta$, where the parameter β is used to control how substantial the decrement value will be.

Candidates with positions $< n$ have their increment calculation according to Equation 5.2, which implements the $\mathfrak{prf}$ function. In this equation, for each position in the rank $eo_{l,rank}^t$ (1 to $n - 1$), the proportion is calculated (Equation 5.1) in order to give higher values for best positions.

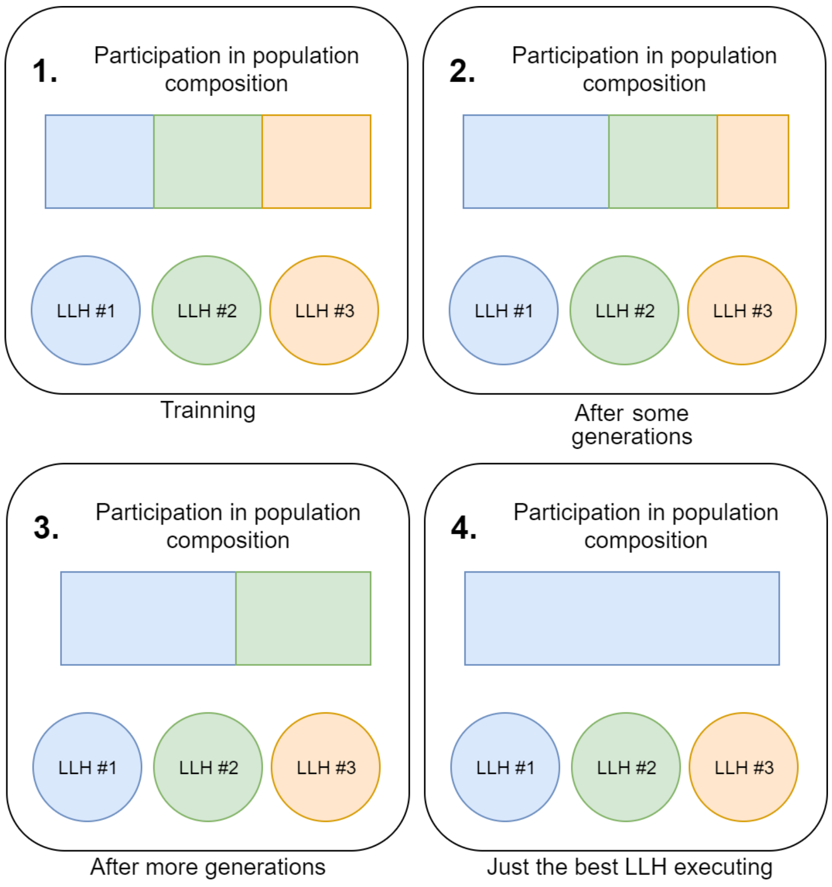

1.
Participation in population composition
LLH #1
LLH #2
LLH #3
Trainning
2.
Participation in population composition
LLH #1
LLH #2
LLH #3
After some generations
3.
Participation in population composition
LLH #1
LLH #2
LLH #3
After more generations
4.
Participation in population composition
LLH #1
LLH #2
LLH #3
Just the best LLH executing

Figure 24: Population generation participation along the execution

$$fx(rank, n) = \left\{ \begin{array}{l} 2^n \text{ if } rank = 1 \\ 2^{n-rank} \text{ otherwise} \end{array} \right\} \tag{5.1}$$

$$pp_i^{t+1} = pp_i^t + \frac{fx(eo_{l,rank}^t, n)}{\sum_{i=1}^n fx(eo_{l,rank}^t, n)} * \beta \tag{5.2}$$

The idea behind these equations is to give more participation to the candidate who won the election, some level of increasing for other candidates according to their positions and decreasing the participation to the less voted candidate. These equations were designed based on empirical studies.

5.3.2 Evaluation and voting

At every δ generations, a new election is started. Here, all *Voter Agents* ($\mathcal{V}$) take the current population from the *Population Artifact*, split it into sub-populations according to which LLH had generated the solution. So, each sub-population $rp \in \mathcal{RP}$ represents a LLH. After this, each *Voter Agent* evaluates all sub-populations according to his multi-objective quality indicator. This generates a set of quality values for each voter. Figure 25 presents this process considering three LLHs as candidates. After evaluating (generating $\mathcal{VP}$), the agents send this information to the *Voting Pool* artifact, performing then a vote.

In MOABHH, tie-breaking between candidates is not coped by the voting method $\mathfrak{election}$, then LLHs can have the same rank after the election.

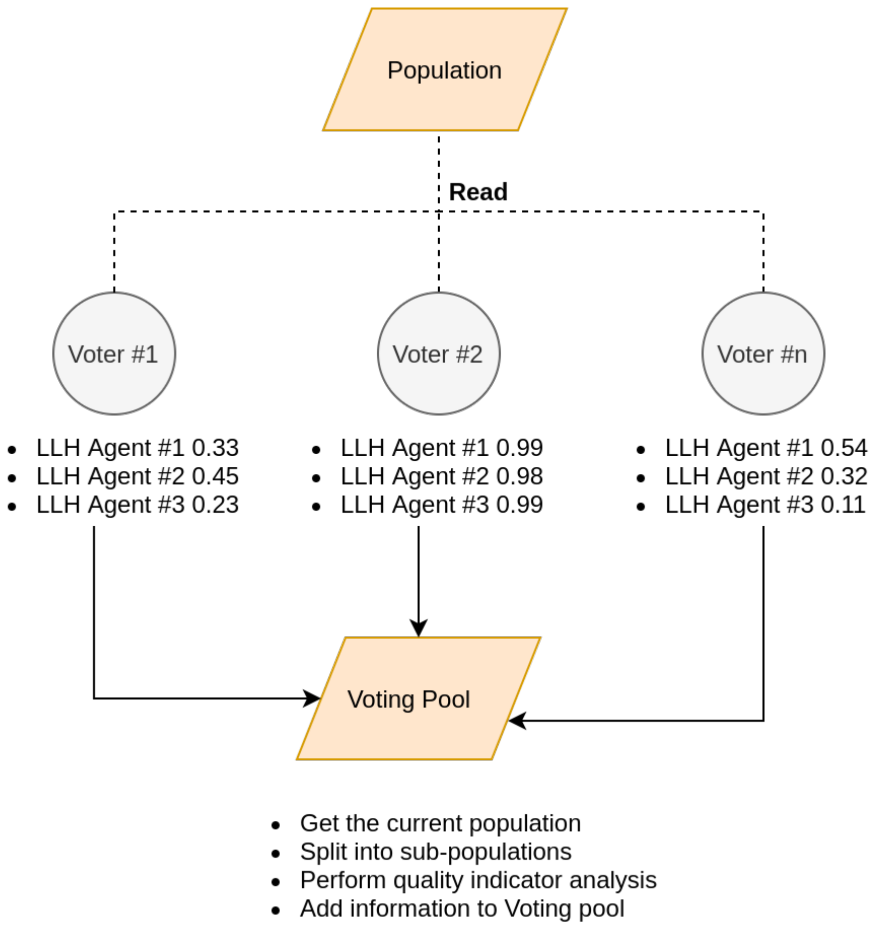


Figure 25: MOABHH evaluation.

After having the voters vote, the *HH Agent* starts to play its role (Figure 26). This agent is responsible for taking all the preferences ($\mathcal{VP}$), and using a voting procedure ($\mathfrak{election}$) in order to create an election outcome ($\mathcal{EO}$). This outcome, as mentioned before, is taken into consideration in the participation assignment. After this, all decision made by the *HH Agent* is stored in the *Decision Space* artifact.

5.3.3 Population allocation

After the *HH Agent* updates participation values in $\mathcal{PP}$, all LLH with participation > 0 are allowed to run for the next δ generations (the $\mathcal{LLH}_{active}$ set). Before running (Figure 27),

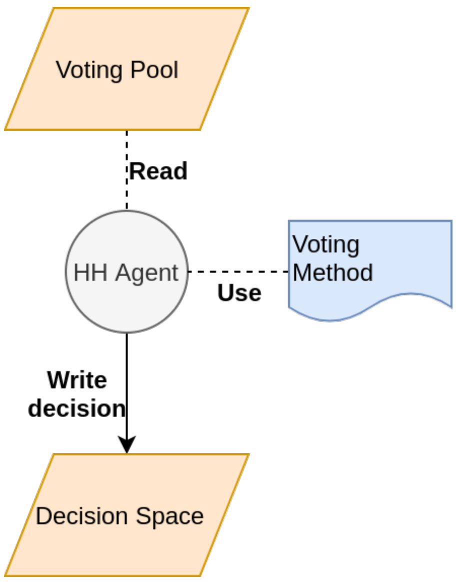


Figure 26: MOABHH evaluation.

each LLH agent read its participation from the *Decision Space* artifact and the whole population from the homonym artifact. During the execution, each i^{th} LLH takes a sub-population by selecting a total of pp_i^t solutions from the current population. The *HH Agent* randomly selects solutions (from the current main population) in order to compose sub-populations, thus bad performing LLHs has more chance to not get stuck trying to generate solutions using sub-optimal solutions. This sub-population is set as the current population in the MOEA instantiated in the LLH agent. So, while the stopping criteria is not reached, that means, while δ generations are not run, the MOEA performs all his evolutionary procedures (i.e., selection, crossover, mutation and elitism). When the stopping criterion is reached, the inner MOEA returns his current population, and then the LLH agent sets the new population to the *Population* artifact. In this process, the surviving solutions are defined by each inner MOEA instance, which means if a LLH agent has to generate ns new solutions, the resulting population also will have ns solutions.

5.3.4 Population sharing

An important aspect of MOABHH is how solutions are shared among the LLH Agents. In fact, this is the aspect that makes this hyper-heuristic also be considered as a cooperative approach. Let's see an example of how a population is shared, considering three agents. First, in Figure 28 the *Problem manager* agent generates the first population of solutions (randomly generated) and then assign it to the *Population Artifact*. In this image, one can observe that

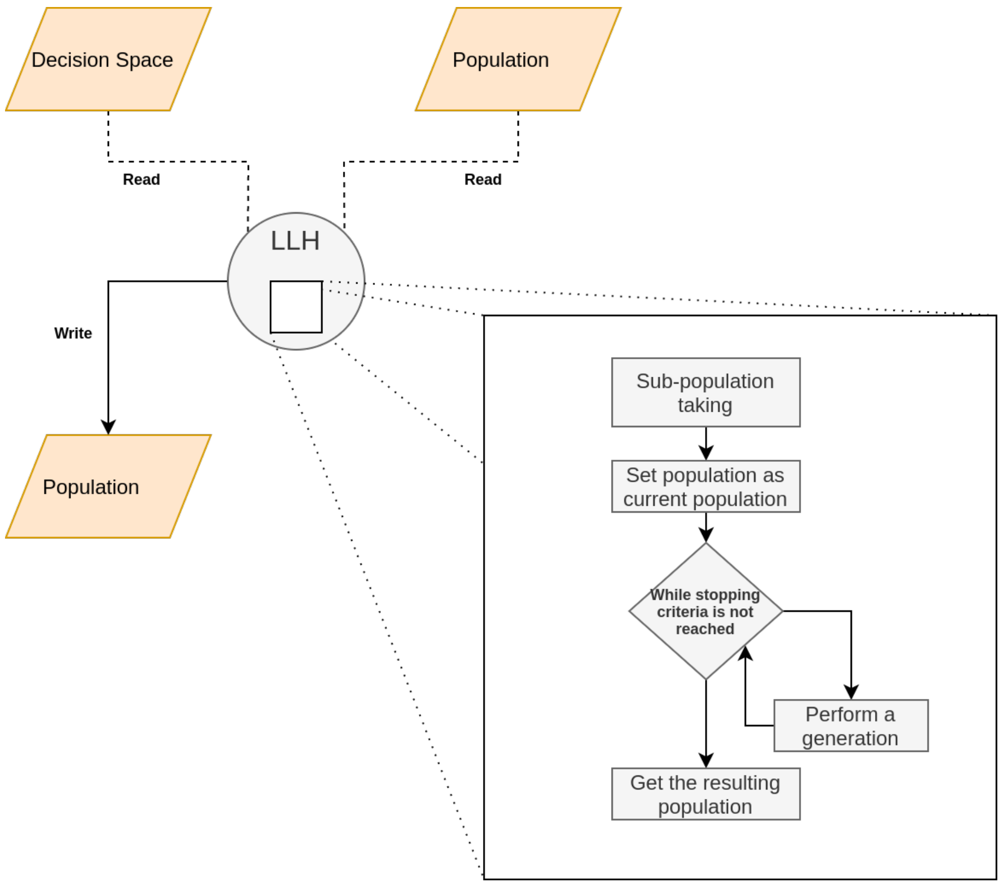

Decision Space
Population
Read
Read
LLH
Write
Population
Sub-population taking
Set population as current population
While stopping criteria is not reached
Perform a generation
Get the resulting population

Figure 27: MOABHH population allocation.

these solutions (inside the big hexagon) are not generated by any *LLH Agent* and for this reason, they are colored as gray here. Then, the *HH Agents* splits this population into three considering the *participation* from *Decision Space Artifact*; initially, all LLHs get a same share, as mentioned in Section 5.3.3. Finally, *LLH Agents* can read their population share from the *Population Artifact*.

LLH Agents execute and each one generates a new population. One can observe, in Figure 29, produced offsprings are colorized according to the color used, in this figure, for each *LLH Agent*. Moreover, some solutions of the previous generations may persist (grey elements). These solutions are stored in the *Population Artifact*, and the *HH Agent* can then join all shares and update the population into the Population Artifact. From this point, now we have solutions from all the *LLH Agents*.

Finally, in Figure 30, almost the same steps performed in Figure 28 are now performed. The difference here lies in the fact that we are not dealing with a random population generated by *Problem Manager* but by *LLH Agents*. Thus, *HH Agent* randomly splits the main population into three. Due to the fact that the population is randomly split, *LLH Agents* can receive

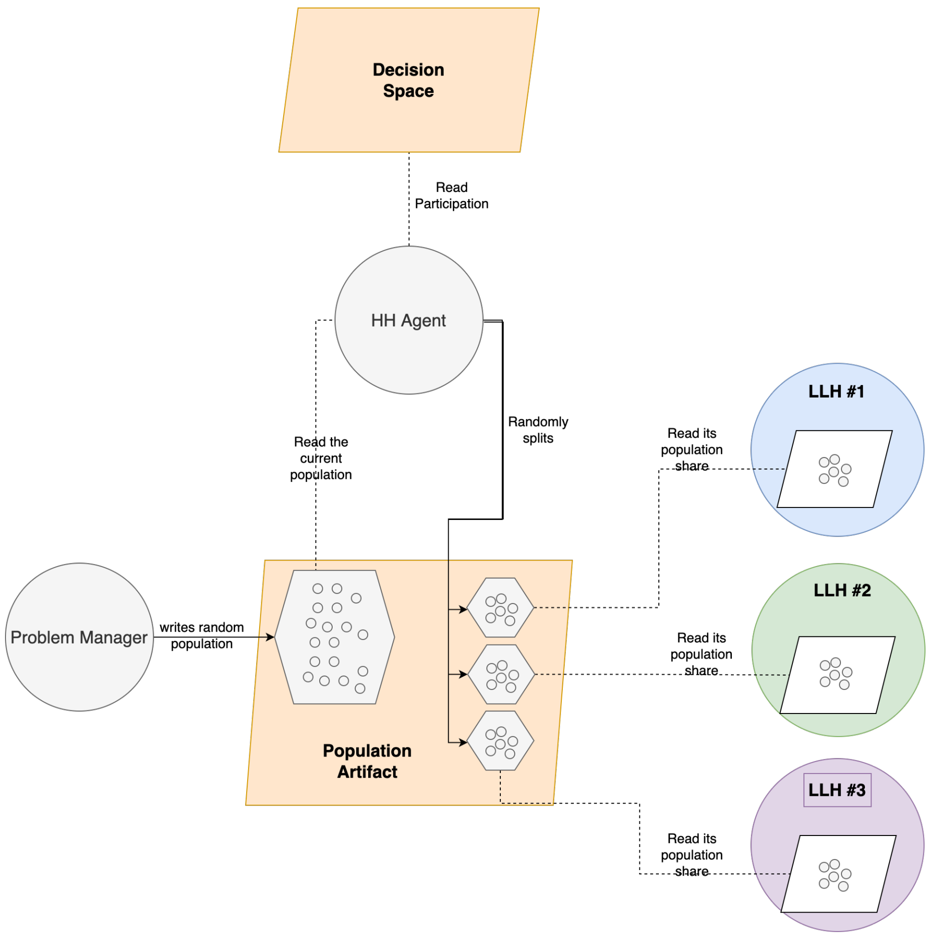


Figure 28: MOABHH population sharing at the first generation.

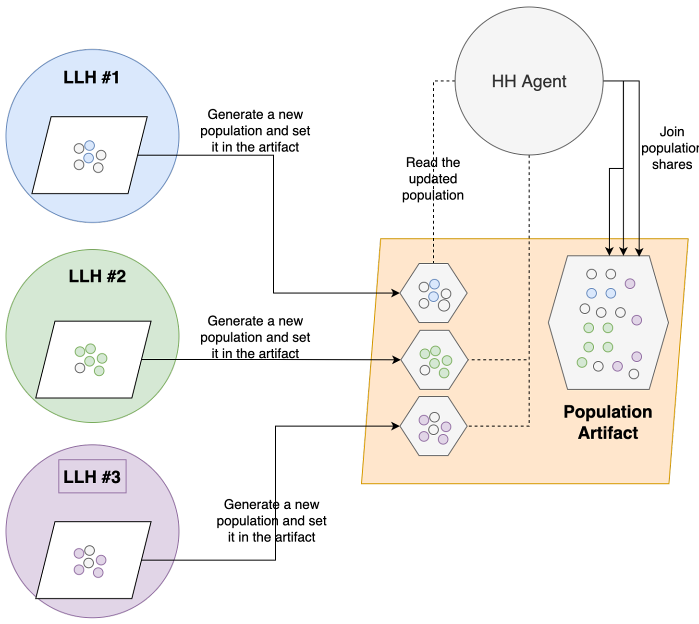

LLH #1
Generate a new population and set it in the artifact
HH Agent
Read the updated population
Join population shares
LLH #2
Generate a new population and set it in the artifact
Population Artifact
LLH #3
Generate a new population and set it in the artifact

Figure 29: MOABHH population generation.

solutions generated by another *LLH Agent*. This is the cooperation aspect of MOABHH: *LLH Agents* compete for participation, but they share solutions with each other. However, MOABHH keeps the relationship between solutions and the LLH that has generated them. That means, solutions will be kept related to a *LLH Agent* that has generated them, even if they are further selected in the next generations by another LLH. Thus, *Voter Agents* will consider this fact when they evaluate *LLH Agents*.

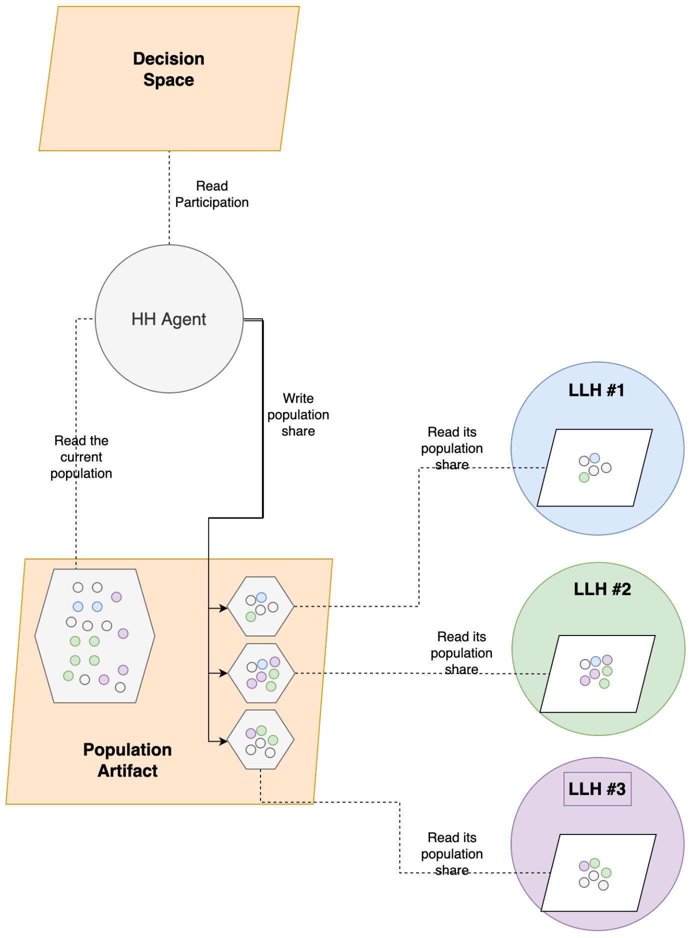

Decision Space
Read Participation
HH Agent
Read the current population
Write population share
Read its population share
LLH #1
LLH #2
Read its population share
LLH #3
Read its population share
Population Artifact

Figure 30: MOABHH population sharing for next generations.

5.4 MOABHH Pseudocode

Algorithm 8 shows the detailed MOABHH steps. First, all agents and artifacts are initialized using MOABHH parameters (Line 3). In this step, the optimization problem is instantiated and all global variables set in *System Variables* artifact. The *Problem manager* agent then generates a random population of solutions (Line 4). The execution continues by allocating equal participation for all low-level heuristic agents (Line 5), so in the beginning, all LLH will generate the same number of solutions during δ generations (Line 8). After that, all LLH returns its surviving population and send it to the *Population Artifact* (Line 9). So, while the stopping criteria are not reached (Line 10), the current population is split into sub-populations according to which LLH generated each solution. These sub-populations are evaluated by the *Voters* in order to generate a ranking of preferences for each $v \in \mathcal{V}$ (Line 11). After this, these preferences are set in the *Voting pool* (Line 12) artifact where the *HH Agent* can read it, perform the voting method $\mathfrak{vm}$ (Line 13), generate the election outcome $\mathcal{EO}^t$ and use it to update the participation table $\mathcal{PP}^t$ (Line 14). At this point, the election winner increases its participation and the losers decrease it. After updating the number of solutions to generate (Line 16), all low-level heuristic agent executes during δ generations (Line 17) and adds new solutions to *Population* artifact (Line 18).

Algorithm 8: MOABHH Pseudocode.

```
1  Input: $p, \mathcal{LLH}, \mathcal{V}, \mathcal{VM}, \delta$
2  begin
3      Initialize agents and artifacts;
4      Initialize the population $\mathcal{RP}$ with random solutions to solve $p$;
5      Uniformily assign participation shares in $\mathcal{PP}$;
6      $\mathcal{LLH}_{active} \leftarrow \mathcal{LLH}$;
7      Consider $\mathcal{PP}$ to assign solutions from $Pop\ \forall llh \in \mathcal{LLH}_{active}$;
8      $\forall llh \in \mathcal{LLH}_{active}$ execute for $\delta$ generation;
9      Update $\mathcal{RP}$ ;
10     while Stopping criteria are not reached do
11         Evaluators in $\mathcal{V}$ evaluate $\mathcal{RP}$ to generate $\mathcal{VP}$;
12         Evaluators $\mathcal{V}$ send preferences $\mathcal{VP}$ to Voting Pool Artifact;
13         HH Agent employs $\mathfrak{election}$, uses $\mathcal{VP}$ to generate the election outcome $\mathcal{EO}$;
14         HH Agent uses $\mathfrak{election}$ to generate $\mathcal{PP}$ considering $\mathcal{EO}$;
15         $\mathcal{LLH}_{active} \leftarrow \mathcal{LLH}$ according to $\mathfrak{active}$;
16         Consider $\mathcal{PP}$ to assign solutions from $\mathcal{RP}\ \forall llh \in \mathcal{LLH}_{active}$;
17         $\forall llh \in \mathcal{LLH}_{active}$ execute for $\delta$ generation;
18         Update the main population $\mathcal{RP}$ ;
19     end
20     return Main population
21 end
```

5.4.1 Architecture execution design

MOABHH is designed to be a *syncronized* multi-agent system, which means agents have to wait for other agents to execute. More precisely, *LLH Agents* execute in parallel, but they have to wait for *HH Agent* to define population shares. *Voter Agents* also execute in parallel but have to wait for *HH Agent* to take resulting population shares and set it in the *Population*

Artifact. *HH Agent* has to wait for other agents to perform their role.

The *Decision Space Artifact* contains information about which agent has to execute. *HH Agent* orchestrate other agents by updating this information.

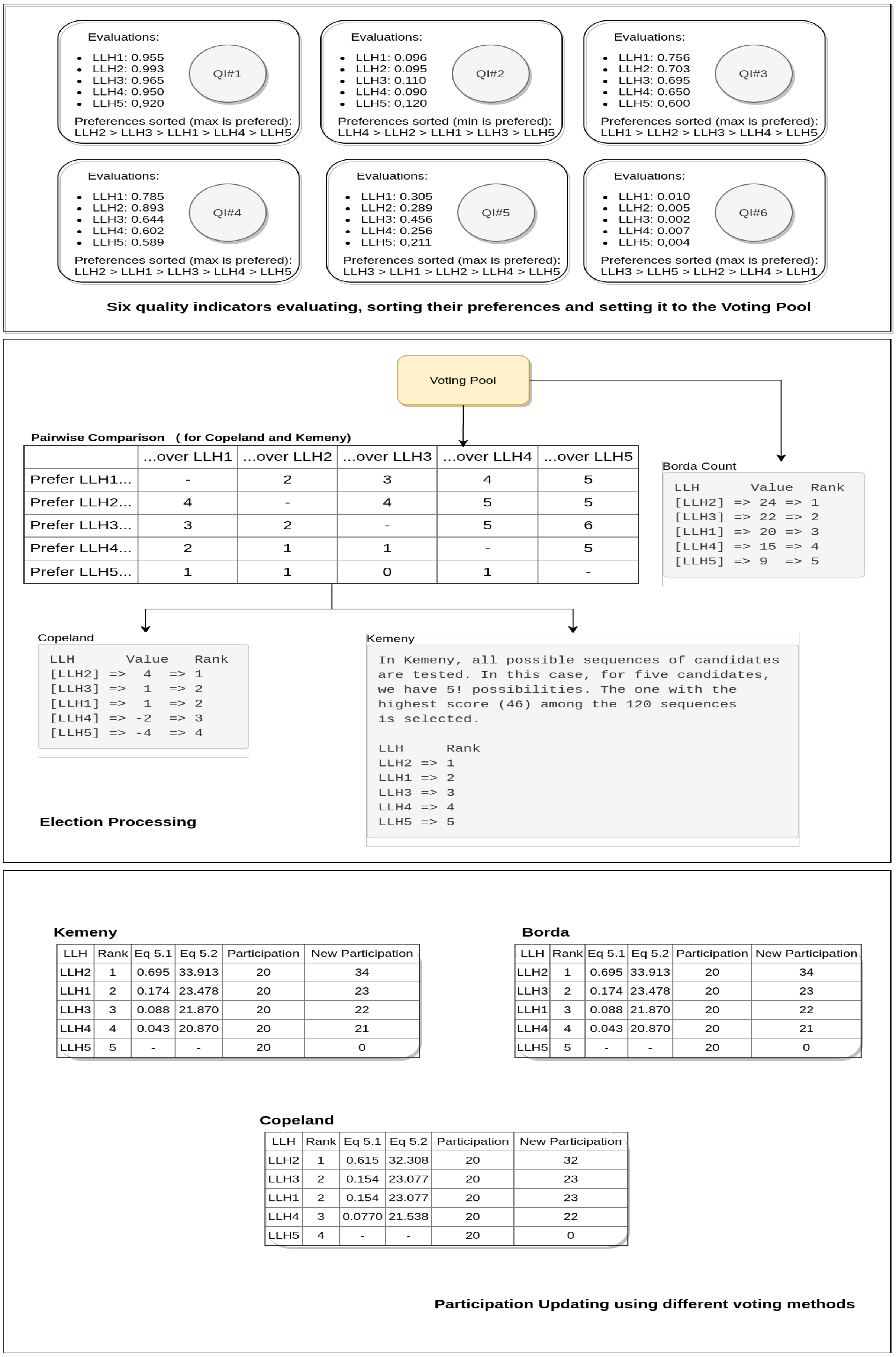
5.5 Example of execution

Figure 31 presents a complete iteration for MOABBH. First, all the five LLH are run in parallel, generating each a new population of solutions. Then, all the resulting populations are evaluated according to all the six voter agents, sorted according to each voter agent's preferences and set to the voting pool artifact. For example, in the Figure 31, the quality indicator #1 (*QI#1*) evaluated *LLH2* as 0.993, while it evaluated *LLH3* with 0.965. Thus, since this quality indicator considers higher values as better, then *LLH2* is preferable than *LLH3*. Quality Indicator #2 (*QI#1*) considers smaller values as better, so it prefers *LLH4* rather than others.

With the evaluation information, MOABHH is able to execute the election. In this example, for didactic purposes, all of the three studied voting methods results are shown. Regarding Borda, the count is directly calculated, providing, for example, a value of 24 for *LLH2* and 15 for *LLH3*. Using this value, one can conclude *LLH2* is the best algorithm, while *LLH4* is the fourth-ranked according to this voting method.

Considering Kemeny and Copeland, the pairwise comparison is first performed using information from the *Voting Pool*, where, for example, *LLH2* is preferable over *LLH1* according to 4 quality indicators. Finally, Copeland runs and assign a value of 4 for *LLH2* (which is also the highest) and 1 to *LLH3* and *LLH1* making this two draw. For Kemeny, all possible combinations of candidate sequences are tested, and the sequence with the highest score (46) is selected. In this sequence *LLH2* is the first-ranked while *LLH3* is the third.

We can see they diverge which one candidate is the second-best: Copeland considers LLH3 and LLH1 tied, Borda considers LLH3, while Kemeny considers LLH1 as the second-best. This also happens for other positions in other election outcome positions. Finally, the election outcomes are taking into consideration by Equations 5.1 and 5.2 and each LLH participation updated. It is worth mentioning these voting methods are never run together since MOABHH can only employ one voting method per time.



Evaluations:

- LLH1: 0.955
- LLH2: 0.993
- LLH3: 0.965
- LLH4: 0.950
- LLH5: 0,920

Preferences sorted (max is prefered):
LLH2 > LLH3 > LLH1 > LLH4 > LLH5

Evaluations:

- LLH1: 0.096
- LLH2: 0.095
- LLH3: 0.110
- LLH4: 0.090
- LLH5: 0,120

Preferences sorted (min is prefered):
LLH4 > LLH2 > LLH1 > LLH3 > LLH5

Evaluations:

- LLH1: 0.756
- LLH2: 0.703
- LLH3: 0.695
- LLH4: 0.650
- LLH5: 0,600

Preferences sorted (max is prefered):
LLH1 > LLH2 > LLH3 > LLH4 > LLH5

Evaluations:

- LLH1: 0.785
- LLH2: 0.893
- LLH3: 0.644
- LLH4: 0.602
- LLH5: 0.589

Preferences sorted (max is prefered):
LLH2 > LLH1 > LLH3 > LLH4 > LLH5

Evaluations:

- LLH1: 0.305
- LLH2: 0.289
- LLH3: 0.456
- LLH4: 0.256
- LLH5: 0,211

Preferences sorted (max is prefered):
LLH3 > LLH1 > LLH2 > LLH4 > LLH5

Evaluations:

- LLH1: 0.010
- LLH2: 0.005
- LLH3: 0.002
- LLH4: 0.007
- LLH5: 0,004

Preferences sorted (max is prefered):
LLH3 > LLH5 > LLH2 > LLH4 > LLH1

Six quality indicators evaluating, sorting their preferences and setting it to the Voting Pool

Pairwise Comparison (for Copeland and Kemeny)

	...over LLH1	...over LLH2	...over LLH3	...over LLH4	...over LLH5
Prefer LLH1...	-	2	3	4	5
Prefer LLH2...	4	-	4	5	5
Prefer LLH3...	3	2	-	5	6
Prefer LLH4...	2	1	1	-	5
Prefer LLH5...	1	1	0	1	-

Borda Count

```
LLH       Value  Rank
[LLH2] => 24 => 1
[LLH3] => 22 => 2
[LLH1] => 20 => 3
[LLH4] => 15 => 4
[LLH5] => 9  => 5
```

Copeland

```
LLH      Value   Rank
[LLH2] =>  4  => 1
[LLH3] =>  1  => 2
[LLH1] =>  1  => 2
[LLH4] => -2  => 3
[LLH5] => -4  => 4
```

Kemeny

```
In Kemeny, all possible sequences of candidates
are tested. In this case, for five candidates,
we have 5! possibilities. The one with the
highest score (46) among the 120 sequences
is selected.

LLH     Rank
LLH2 => 1
LLH1 => 2
LLH3 => 3
LLH4 => 4
LLH5 => 5
```

Election Processing

Kemeny

LLH	Rank	Eq 5.1	Eq 5.2	Participation	New Participation
LLH2	1	0.695	33.913	20	34
LLH1	2	0.174	23.478	20	23
LLH3	3	0.088	21.870	20	22
LLH4	4	0.043	20.870	20	21
LLH5	5	-	-	20	0

Borda

LLH	Rank	Eq 5.1	Eq 5.2	Participation	New Participation
LLH2	1	0.695	33.913	20	34
LLH3	2	0.174	23.478	20	23
LLH1	3	0.088	21.870	20	22
LLH4	4	0.043	20.870	20	21
LLH5	5	-	-	20	0

Copeland

LLH	Rank	Eq 5.1	Eq 5.2	Participation	New Participation
LLH2	1	0.615	32.308	20	32
LLH3	2	0.154	23.077	20	23
LLH1	2	0.154	23.077	20	23
LLH4	3	0.0770	21.538	20	22
LLH5	4	-	-	20	0

Participation Updating using different voting methods

Figure 31: MOABHH Example.

5.6 Conclusion

This chapter detailed this thesis proposal detailing the model and the architecture. The next chapter presents the related work and details about state-of-art algorithms considered in this thesis.

6 RELATED WORK

Applying meta-heuristics is not a trivial task, and over the years, several attempts have been proposed to diminish the effort on choosing and configuring an algorithm. Here we can cite two approaches: (i) parameter control and (ii) hyper-heuristics.

The parameter control (KARAFOTIAS; HOOGENDOORN; EIBEN, 2015b) approch focuses on diminishing the effort on setting up parameters by automatically setting them. These work usually employ fixed heuristics and focus on controlling their parameters (KARAFOTIAS; HOOGENDOORN; EIBEN, 2015b). For example, controlling mutation rates (VAFAEE; NELSON, 2010; BACK, 1992), crossover rates (SRINIVAS; PATNAIK, 1994) and population sizes (SMORODKINA; TAURITZ, 2007; EIBEN; MARCHIORI; VALKÓ, 2004; HARIK; LOBO, 1999; HINTERDING; MICHALEWICZ; PEACHEY, 1996) for genetic algorithms. There are also some work with focus on controlling multiple parameters (EIBEN et al., 2007; KARAFOTIAS; HOOGENDOORN; EIBEN, 2015a) such as mutation rate, crossover rate, the population size and size of the selection tournament.

Research from parameter control will not be detailed in this chapter since they are not directly related to this thesis. Concerning hyper-heuristic, after analyzing this subject literature, we can classify the proposed approaches, according to which component is employed as low-level-heuristic, in two groups: (1) Heuristic as LLH (2) Meta-heuristics as LLH. Both these groups can also be split into agent-based and non-agent-based approaches, as presented next.

A preliminary literature review was published in (CARVALHO; SICHMAN, 2019).

6.1 Group 1: Heuristics as LLH

The first group has the majority of the research in the literature (MAASHI; ÖZCAN; KENDALL, 2014; DRAKE et al., 2019). As mentioned above, they are here classified as agent-based and non-agent-based.

6.1.1 Agent-based approaches

Considering agent-based approaches, Ouelhadj and Petrovic (OUELHADJ; PETROVIC, 2010) employed search operators, such as Swap, Inversion, Insertion, and Permutation, as LLH agents to solve Permutation Flow Shop, a single-objective discrete optimization problem. This is a cooperative hyper-heuristic, where the heuristic agents perform a local search through the same solution space starting from the same or different initial solution and using different low-level heuristics. The agents exchanges their best solutions (according to the objective function). After a generation, the best solutions are selected from all agents. This approach performs a greedy selection strategy to select an LLH to execute.

Meignan et al. (MEIGNAN; KOUKAM; CRÉPUT, 2010) propose a selection hyper-heuristic where agents are responsible for concurrently explore the search space of an optimization problem in a cooperative way, where agents organized in a coalition cooperate by the exchanging of information about the search space and their experiences in order to improve agents behaviors. To perform the search, an agent uses several heuristics, which are scheduled by an adaptive decision process, based on heuristic rules adapted along the optimization process by individual learning. In this approach, a search agent keeps three solutions: the current, the best-found solution of the agent and the best solution of the entire coalition, and it can employ several operators on its current solution. This approach was applied to solve the Vehicle Routing Problem (VRP). In this work, agents cooperate intensively due to the use of coalitions. Moreover, each agent had a set of heuristics differently from the one used in Ouelhadj and Petrovic's work.

6.1.2 Non-Agent-based approaches

In this section, non-agent-based approaches are explained, especially research that treated heuristics or algorithm components (no complete algorithm instances) as low-level-heuristics.

In (COWLING; KENDALL; SOUBEIGA, 2001), the authors proposed the *Choice Function*, an equation responsible for rank heuristics considering the algorithm performance, in this case, a fitness function for a mono-objective problem, and the computational time. This approach was applied to the sales summit problem (the problem of matching suppliers to customers). The hyper-heuristic was responsible for controlling ten operations (LLHs), including: removing a delegate from a meeting with a particular supplier, adding a delegate to a supplier to allow them to meet and remove meetings from a supplier who has more than their allocation.

In (MCCLYMONT et al., 2013), the authors presented a selection hyper-heuristic based on Markov chains and Reinforcement Learning to select LLHs from a set of four low-level mutational heuristics specialized in the optimization of water distribution network. In this approach, NSGA-II and SPEA2 were employed as the MOEAs. For the performance evaluation, the authors employed the ratio of dominating solutions produced by each LLH.

In (LI, K. et al, 2014), the FFRMAB was proposed, a variation of the original Multi-Armed Bandit (AUER; CESA-BIANCHI; FISCHER, 2002) (MAB) where the *Fitness Rate Ranking (FRR)* was proposed as reward assignment. In this hyper-heuristic, a set of Differential Evolution operators was considered as LLH for being chosen to a fixed instance of MOEA/D. Following the choice of DE operators, the work of (GONÇALVES et al., 2015) proposed a hyper-heuristic based on the choice function to control a set of five Differential Evolution operators for MOEA/D. Results showed this hyper-heuristic overcoming the performance of the standard MOEA/D (using a single operator) when solving ten unconstrained benchmark functions with two and three objectives. In (GONÇALVES; ALMEIDA; POZO, 2015), the authors applied a similar approach using several versions of MAB instead of using a Choice Function. In this case, the CEC 2009 benchmark was employed for performance evaluation. In (ALMEIDA et al., 2020) the authors evaluated three different versions of MAB by applying them to the permutation flow shop problem. In this HH, operators were chosen for MOEA/DD (LI et al., 2015b).

In (GUIZZO et al., 2015), the authors designed a hyper-heuristic to solve the Class Integration Test Order Problem (aO et al., 2014), a software engineering problem where nodes from a graph have to be visited, where these nodes are the classes to be tested. This hyper-heuristic was built using a choice function (MAASHI; ÖZCAN; KENDALL, 2014) and a Multi-Armed Bandit (AUER; CESA-BIANCHI; FISCHER, 2002) to select a LLH from a set of nine to operate together with a fixed MOEA, in this case, the NSGA-II algorithm. This set was built by combining different crossover and mutation operators. The evaluation of LLHs was performed based on the dominance relationship among parent solutions and their offspring. In (GUIZZO; VERGILIO; POZO, 2015), this approach was tested considering SPEA2 as the fixed MOEA. In (CARVALHO, 2015), the author tacked the same problem by creating a hyper-heuristic based on FRRMAB (LI, K. et al, 2014) and considering the same set of LLHs. In this case, the Multi-objective Evolutionary Algorithm Based on Decomposition (ZHANG;

LI, 2007) (MOEA/D) was the fixed algorithm. Among all these versions, the Choice Function applied together with NSGA-II ((GUIZZO et al., 2015) version) got the best results.

In (CASTRO; POZO, 2015), a hyper-heuristic focused on selecting the leader and archiving method for a multi-objective Particle Swarm Optimization. In this approach, a variant of the Choice function considers the R2 performance indicator for selecting components from two archiving strategies in combination with three leader selection methods. The authors tested this approach using a set of DTLZ benchmark functions with dimensions varying from 2 to 20 and presented competitive results.

In (KHEIRI; KEEDWELL, 2017), a sequence-based selection hyper-heuristic was designed using a hidden Markov model for solving a suite of high school timetabling problems. This approach selects LLHs considering previous invoked LLHs performance and then learning effective sequences of LLHs. The approach updates the scores using a Reinforcement Learning procedure.

6.2 Group 2: Meta-heuristisc as LLH

The second group is where this thesis takes place. Their components are here also classified as agent-based and non-agent-based.

6.2.1 Agent-based approaches

Cadenas et al. (CADENAS; GARRIDO; MUNOZ, 2007) introduced a cooperative multi-agent meta-heuristic approach, where agents communicate their best solutions using a common blackboard. This blackboard is monitored by a coordinator agent who is responsible for modifying meta-heuristic agent behaviors based on fuzzy rules, which take into account algorithm performance in the search. The authors tested their approach using 0/1 knapsack problems.

Malek (MALEK, 2010) proposes a multi-agent hyper-heuristic to solve several combinatorial problems by considering the Genetic Algorithm, Tabu Search, Simulated Annealing, Particle Swarm Optimization, Ant Colony Optimization as algorithm agents. In this approach, there is also a Problem agent, a Solution Pool agent (responsible for keeping all solutions), and an Adviser agent, an agent who provides parameter settings for the algorithms and receives reports from them. All algorithm agents of the same kind are associated with a common

Adviser agent.

Acan and Lotfi (ACAN; LOTFI, 2016) propose a collaborative hyper-heuristic architecture designed for multi-objective continuous optimization problems. In their approach, the population of solutions is split into sub-populations based on Pareto dominance, and then these sub-populations are assigned each one to a meta-heuristic agent, based on a cyclic or round-robin order, making each meta-heuristics agent operate on a sub-population in subsequent sessions. Meta-heuristic agents have their own population of non-dominated solutions extracted in a session, while there is also a global population of solutions keeping all non-dominated solutions found in the search. This study set MOGA, NSGAII, SPEA2, MODE, IMOPSO, AMOSA as meta-heuristic agents, and it was evaluated considering the CEC2009 benchmark.

Nugraheni and Abednego (NUGRAHENI; ABEDNEGO, 2016) propose an approach to select one of three agent Hyper-heuristics based on Genetic Programming (GPHH agent), Genetic Algorithm Hyper-Heuristic (GAHH agent), and Simulated Annealing Hyper-Heuristics (SAHH agent). These HH agents choose some low-level heuristics and work in the search space of heuristics rather than a space of solutions directly.

Martin et al. (MARTIN et al., 2016) propose a multi-agent hyper-heuristic where each agent implements a different meta-heuristic/local search combination. These agents also adapt themselves along with the search by using a proposed cooperation protocol based on reinforcement learning and pattern matching. Two kinds of agents are employed: launcher and meta-heuristic agents. The launcher is responsible for instantiating and keep the optimization problem, set up algorithms, and create initial solutions. The meta-heuristic agent contains an algorithm responsible for searching collectively for good quality solutions. The authors evaluated their approach using Permutation Flow Shop, CVRP and Nurse Rostering mono-objective problems, while employing meta-heuristic agents.

6.2.2 Non-Agent-based approaches

In (VRUGT; ROBINSON, 2007), the authors proposed AMALGAM (a multi-algorithm genetically adaptive multi-objective), a cooperative search approach where meta-heuristics run simultaneously for generating offspring from a population share. This approach work as following: First, a random initial population is generated, then each algorithm generates an offspring with a predefined size. AMALGAM then takes all the generated solutions and compares them with the previous population in order to generate the current population of solutions. The participation in generating new solutions is updated according to the number of surviving solutions from each MOEA to in the mating pool. There are some similarities between this thesis and ALMAGAM. Both consider parallel algorithms generating solutions from a shared population. The difference between them lies in the fact ALMAGAM by itself performs the elitism procedure (decides which solutions should survive in the next population), while this thesis approach lets each meta-heuristic perform their elitism. Another difference is how they compute the participation. In this thesis approach Social Choice Theory is employed to summarize several quality indicators evaluations, while in ALMAGAM, only one criterion is considered: the Pareto Dominance.

In (VÁZQUEZ-RODRÍGUEZ; PETROVIC, 2012), the authors combined a genetic algorithm with a mixture experiment to create a hyper-heuristic. A mixture experiment is a design of experiments technique that allows to exploit accumulated knowledge efficiently and to express it as a probability (CORNELL, 2011). This hyper-heuristic selects solutions to compose the main population applying four different selection criteria. Each criterion was used considering an associated probability based on its performance, calculated during the search.

In terms of performance, the three following works are considered the state-of-art for selecting the most suitable multi-objective algorithm, specially HHRL. Because of that, these works will be detailed in the sequence.

6.2.2.1 HHCF

Maashi et al. (MAASHI; ÖZCAN; KENDALL, 2014) proposed an online selection hyper-heuristic based on the Choice Function (COWLING; KENDALL; SOUBEIGA, 2001) named Hyper-Heuristic based on Choice Function. Their work aimed to select, one at a time, a LLH from a set H (with size n), and apply it along with g generations. To evaluate the performance of the LLHs, this approach uses a two-level-ranking system. First, each LLH is evaluated according to a group of m quality indicators, here compose by Hypervolume, RNI, UD, and AE. At this point, a table containing all the quality indicators values for each LLH is created, a table with size $4 * n$. A second table ($Freq_{rank}$) is generated by computing how many times each LLH has the best value for each quality indicator.

In order to select which LLHs to execute, HHCF uses Equation 6.1, which is composed by an intensification term f_1 and an exploration term f_2 weighted by a parameter α.

$$F(h_i) = \alpha f_1(h_i) + f_2(h_i) \tag{6.1}$$

The intensification term f_1 is calculated by Equation 6.2. In this equation, n is the number of low-level heuristics, $Freq_{rank}(h_i)$ is the number of times that a given low-level heuristic h is the best one according to all quality indicators, $RNI_{rank}(h_i)$ is the rank of the low-level heuristic according to RNI quality indicator.

$$f_1(h_i) = 2 * (n + 1) - (Freq_{rank}(h_i) + RNI_{rank}(h_i)) \tag{6.2}$$

The exploration term f_2 is the computational waiting time (WT) that a given algorithm has waited inactive. Due to the different computational effort demanded by the problems, f_2 is normalized by Equation 6.3.

$$f_2(h_i) = \frac{WT_{h_i}}{\sum_{j=0}^{n} WT_j} * 100 \tag{6.3}$$

Algorithm 9 illustrates how HHCF works. First, a random population of solutions is generated (Line 7) and used in the initialization process (Line 8). In this process, each $h \in H$ executes for g generations (Line 9), the current population Pop is updated and the value of quality indicators for Pop are computed and stored (Line 10). Afterward, the algorithm continues with the process until the stopping criteria are met, by ranking each $h \in H$ (Line 1 2) and calculating Eq. 6.2 (Line 13) and Eq. 6.3 (Line 14). With this information, the LLH which maximizes Eq. 6.1 is selected (Line 15) and used to generate solutions during g generations (Line 16). Finally, the new population Pop' is created using Pop and the offspring population (Line 17), and all the quality indicators are recalculated for h_i using Pop' (Line 18).

Figure 32 presents an example of how HHCF works. Every time a LLH is run, the quality indicators are calculated considering the resulting population of solutions generated by the LLH. Hence, HHCF associates quality indicator values to LLHs. HHCF selects the next LLH to run after performing the two-ranking system and calculating the equations for all LLHs.

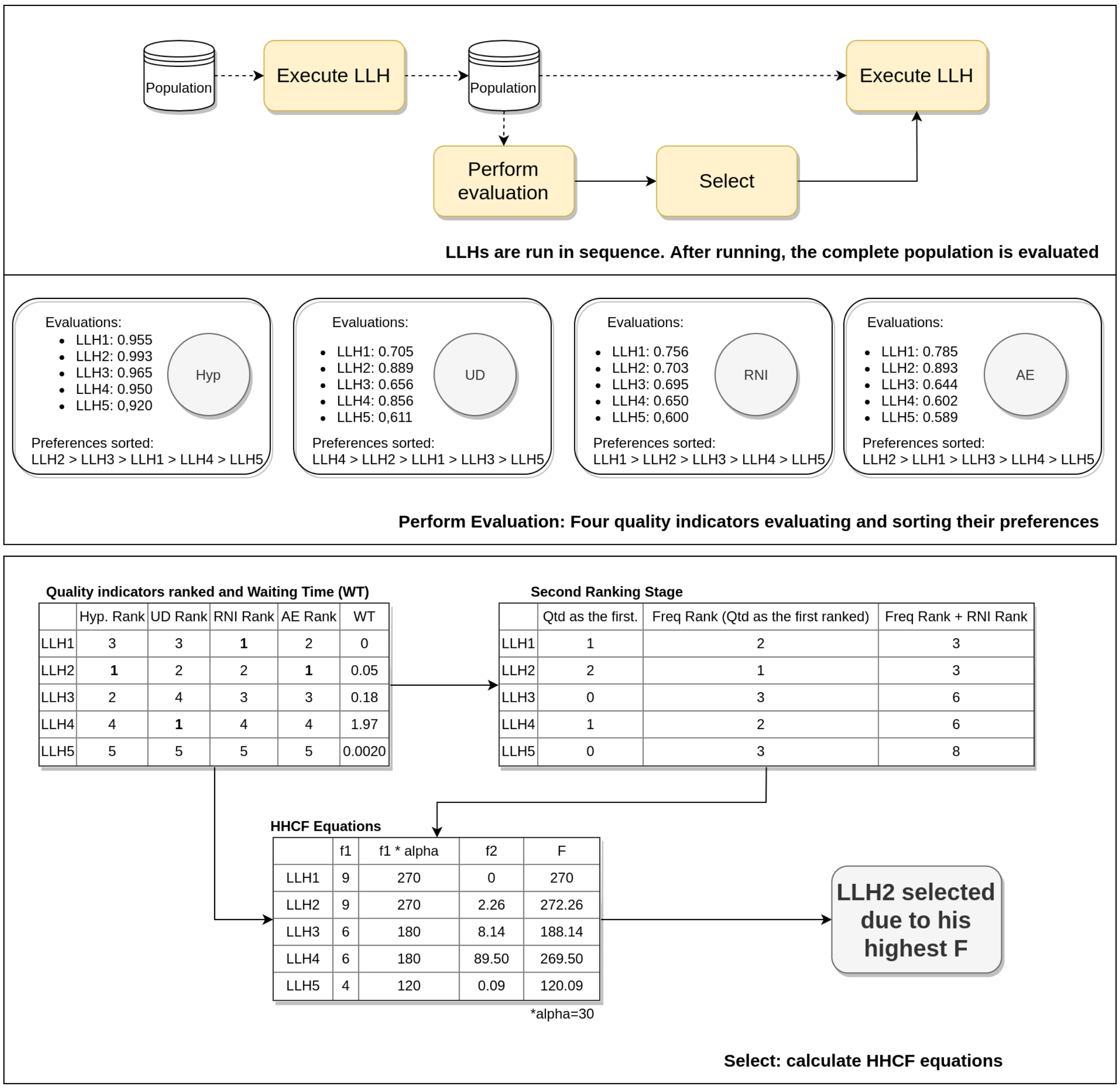
Algorithm 9: HHCF Pseudocode.

```
1  Input:
2  Problem;
3  $g$ - generations before evaluate an LLH;
4  $H$: set of LLHs $\{h_1, ..., h_i, ..., h_n\}$;
5  $\alpha$ intensification parameter;
6  begin
7      Generate a random population of solutions $Pop$;
8      Initialize components using $H$;
9      All $h \in H$ uses $Pop$ to generate $Pop'$ during $g$ generations;
10     Compute all quality indicators for all $h \in H$;
11     while A stopping criterion is not reached do
12         Compute $Freq_{rank}$ and $RNI_{rank}$ for all $h \in H$;
13         Equation 6.2 is computed for all $h \in H$;
14         Equation 6.3 is computed for all $h \in H$;
15         Select $h_i$ according to Eq 6.1;
16         $h_i$ executes for $g$ generations and generates $Pop'$;
17         $Pop \leftarrow Pop'$ //acceptance criterion;
18         Compute all quality indicators for $h_i$;
19     end
20     return $Pop$
21 end
```

In this example, LLH1 is the best according to RNI indicator, LLH2 is the best according to Hypervolume and AE, LLH4 is the best for UD, and the others do not achieve the best position for any quality indicator. This information is considered to generate a second table. In this table, the number of times as the first is counted and sorted. $Freq_{rank}$ is then obtained and used to calculate *RNI Rank* + $Freq_{rank}$. After generating both tables, equations 6.1, 6.3 and 6.2 can be computed and the LLH with the highest F selected. For example, LLH2 has the highest F value with 272.26 in Figure 32.

In the first study, HHCF was tested using the WFG benchmark and the Crashworthiness problem. The authors used the algorithms NSGAII, SPEA2 and MOGA (FONSECA; FLEMING, 1998) as LLHs, and compared their results to Amalgam and to all single MOEAs. The experimental results indicated the success of HHCF outperforming them all.

In (CARVALHO; VERGILIO; POZO, 2015), the authors evaluated HHCF replacing MOGA by IBEA and evaluated the approach on solving another real-world problem: Class Integration Test Order Problem. In (LI; ÖZCAN; JOHN, 2017), HHCF was also applied to another real-world problem: the Wind Farm layout optimization (TRAN et al., 2013) also considering IBEA, SPEA2 and NSGAII as LLHs and having competitive results. Although this hyper-heuristic



Quality indicators ranked and Waiting Time (WT)

	Hyp. Rank	UD Rank	RNI Rank	AE Rank	WT
LLH1	3	3	**1**	2	0
LLH2	**1**	2	2	**1**	0.05
LLH3	2	4	3	3	0.18
LLH4	4	**1**	4	4	1.97
LLH5	5	5	5	5	0.0020

Second Ranking Stage

	Qtd as the first.	Freq Rank (Qtd as the first ranked)	Freq Rank + RNI Rank
LLH1	1	2	3
LLH2	2	1	3
LLH3	0	3	6
LLH4	1	2	6
LLH5	0	3	8

HHCF Equations

	f1	f1 * alpha	f2	F
LLH1	9	270	0	270
LLH2	9	270	2.26	272.26
LLH3	6	180	8.14	188.14
LLH4	6	180	89.50	269.50
LLH5	4	120	0.09	120.09

Figure 32: HHCF Example.

yielded good results, the use of a two-level ranking approach was not appropriately justified and there was no theoretical background to it. However, it provided an interesting view of considering multiple multi-objective quality indicators, and an inspiration for further work and for this thesis.

6.2.2.2 HHLA and HHRL

Learning Automata-based Multi-Objective Hyper-Heuristic with a Ranking scheme Initialization (Li; ÖZCAN; John, 2019) implements a learning automata whose action is to select a LLH at each decision point, while the optimization problem is solved. There are two versions available of this algorithm: HHLA and HHRL. The only difference resides in the fact that HHRL employs an initialization process used in order to reduce the number of LLH in the pool.

HHLA and HHRL employ machine learning techniques, in special *Q-Learning* in order to identify which is the best action to take, in this case, which LLH has to be selected in a given moment. For this purpose, the Hypervolume improvement is calculated considering the current (j) and the previous values (i). This is performed as Equation 6.4.

$$v_j = \frac{hyp_j}{hyp_j - hyp_i} \tag{6.4}$$

The *reward* is then calculated according to Equation 6.5 taking into account the improvement.

$$r_{(i,j)} = v_j(t) - v_i(t-1) \tag{6.5}$$

The *Q-Table* then can be updated according to Equation 6.6 which considers the reward $r_{(i,j)}$, current *Q-Values* and a α parameter.

$$Q_{(i,j)}(k+1) = Q_{(i,j)}(k) + \alpha[r_{(i,j)}(k+1) - Q_{(i,j)}(k)] \tag{6.6}$$

Before updating the transitioning probability table P, these hyper-heuristics employ Equation 6.7, where the *Q-Value* is multiplied by a control parameter named m and sum with 0.1.

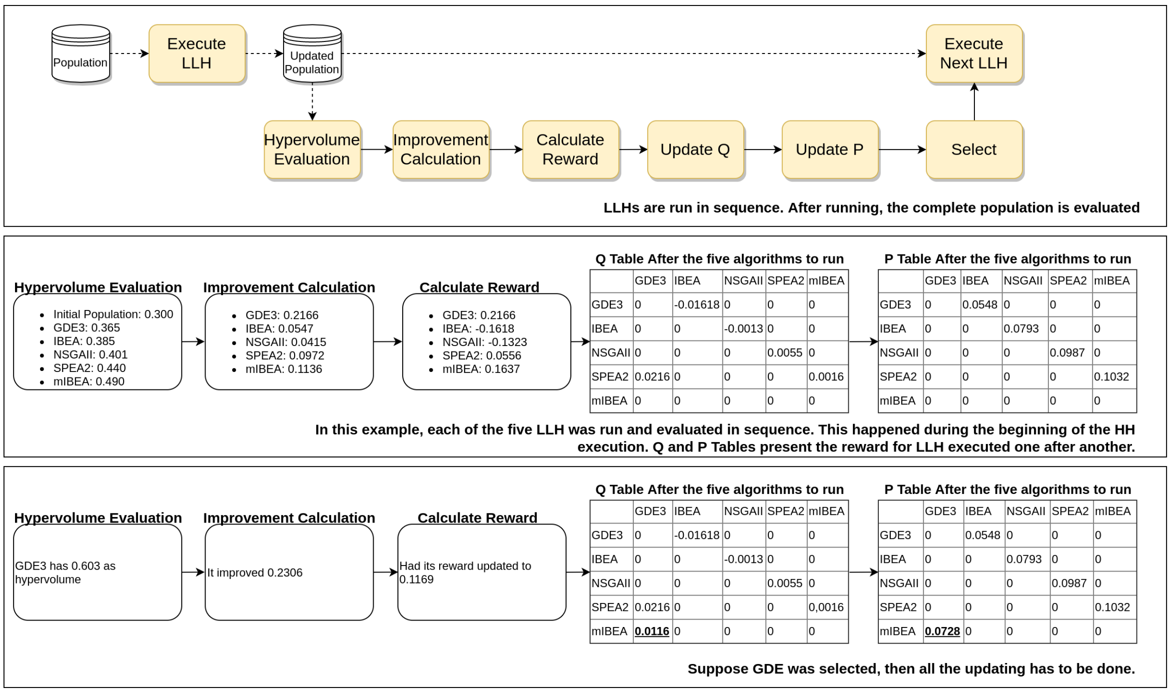
$$\lambda_{(i,j)}(t) = 0.1 + mQ_{(i,j)}(k+1) \tag{6.7}$$

Finally, the transitioning table is updated. This is performed to all the LLHs (actions). In case the LLH was the last chosen, then Equation 6.8 updates P. Otherwise, this is performed by Equation 6.9.

$$p_{(i,j)}(t+1) = p_{(i,j)}(t) + \lambda_{(i,j)}(t)\beta(t)(1 - p_{(i,j)}(t)) - \lambda_{(i,j)}(t)(1 - \beta(t))p_{(i,j)}(t) \tag{6.8}$$

$$p_{(i,j)}(t+1) = p_{(i,j)}(t) - \lambda_{(i,j)}(t)\beta(t)p_{(i,j)}(t) + \lambda_{(i,j)}(t)(1 - \beta(t))\left[\frac{1}{r-1} - p_{(i,j)}(t)\right] \tag{6.9}$$

Figure 33 presents an example of an application of how HHRL/HHLA learns from the hypervolume calculating step towards the *P-Table* updating. First, all five LLHs are run in sequence in order to get the first feedback on how each LLH performs. After executing all of them, the selection procedure (explained later) is run, and GDE3 is selected. Thus all the updating processes are run for updating the action GDE3 being selected after mIBEA.



LLHs are run in sequence. After running, the complete population is evaluated

Hypervolume Evaluation

- Initial Population: 0.300
- GDE3: 0.365
- IBEA: 0.385
- NSGAII: 0.401
- SPEA2: 0.440
- mIBEA: 0.490

Improvement Calculation

- GDE3: 0.2166
- IBEA: 0.0547
- NSGAII: 0.0415
- SPEA2: 0.0972
- mIBEA: 0.1136

Calculate Reward

- GDE3: 0.2166
- IBEA: -0.1618
- NSGAII: -0.1323
- SPEA2: 0.0556
- mIBEA: 0.1637

Q Table After the five algorithms to run

	GDE3	IBEA	NSGAII	SPEA2	mIBEA
GDE3	0	-0.01618	0	0	0
IBEA	0	0	-0.0013	0	0
NSGAII	0	0	0	0.0055	0
SPEA2	0.0216	0	0	0	0.0016
mIBEA	0	0	0	0	0

P Table After the five algorithms to run

	GDE3	IBEA	NSGAII	SPEA2	mIBEA
GDE3	0	0.0548	0	0	0
IBEA	0	0	0.0793	0	0
NSGAII	0	0	0	0.0987	0
SPEA2	0	0	0	0	0.1032
mIBEA	0	0	0	0	0

In this example, each of the five LLH was run and evaluated in sequence. This happened during the beginning of the HH execution. Q and P Tables present the reward for LLH executed one after another.

Hypervolume Evaluation

GDE3 has 0.603 as hypervolume

Improvement Calculation

It improved 0.2306

Calculate Reward

Had its reward updated to 0.1169

Q Table After the five algorithms to run

	GDE3	IBEA	NSGAII	SPEA2	mIBEA
GDE3	0	-0.01618	0	0	0
IBEA	0	0	-0.0013	0	0
NSGAII	0	0	0	0.0055	0
SPEA2	0.0216	0	0	0	0,0016
mIBEA	**0.0116**	0	0	0	0

P Table After the five algorithms to run

	GDE3	IBEA	NSGAII	SPEA2	mIBEA
GDE3	0	0.0548	0	0	0
IBEA	0	0	0.0793	0	0
NSGAII	0	0	0	0.0987	0
SPEA2	0	0	0	0	0.1032
mIBEA	**0.0728**	0	0	0	0

Suppose GDE was selected, then all the updating has to be done.

Figure 33: HHRL and HHLA Example.

Algorithm 10 illustrates how both hyper-heuristics work. First, all the initialization process is performed (Line 6). The algorithm continues while a stopping criterion is not reached, this hyper-heuristic apply the current LLH h_i to the current population (Pop) during g generations producing a new offspring (Pop'). In the following, Pop and Pop' are combined to generate the new current population Pop. In Line 11, this HH verifies whether it is time to switch or not to another LL: this is performed by verifying if there is an improvement in the Hypervolume value (compared to previous iterations). If it is the case, the current LLH keeps running, and if not, the reinforcement learning scheme updates the transition matrix P. Finally, another LLH is selected by the $\varepsilon-$RoulleteGreedy method from A, considering the transition matrix P.

Algorithm 10: HHLA and HHRL Pseudocode, adapted from (Li; ÖZCAN; John, 2019)

```
1  Input:
2  Problem;
3  $H$: set of LLHs $\{h_1, ..., h_i, ..., h_n\}$;
4  $g$ fixed number of generations;
5  begin
6      $[A, P, Pop, h_i] \leftarrow Initialization(H)$;
7      while A stopping criterion is not reached do
8          $Pop' \leftarrow$ ApplyMetaHeuristic($h_i$,$Pop$,$g$ );
9          $Pop \leftarrow Pop'$ //acceptance criterion;
10         //decide whether to switch to another metaheuristic;
11         if switch() then
12             LearningAutomataUpdateScheme($P$);
13             $h_i \leftarrow$ SelectMetaheuristic(P, A);
14         end
15     end
16     return $Pop$
17 end
```

The $\varepsilon-$RoulleteGreedy method, proposed in this work, focuses on exploring different transition pairs by performing a given number of trials in order to get a better view of LLH pairwise performance at the early stage. Then it becomes more and more greedy exploiting the accumulated knowledge.

As mentioned before, the only difference between HHLA and HHRL lies in the initialization method. Algorithm 11 describes this process. First, the method creates a random population of solutions (Line 5) and the transition matrix P (Line 6), which describes the selection probabilities of transitions between LLHs. If HHLA is being run (Line 7), all LLHs are allowed to execute. Otherwise, only allowed LLH is selected.

For this purpose, the set of LLHs (H) is reduced in order to eliminate poor-performing LLHs. This works as follows: First, all LLH are executed in sequence for a number of stages. Every time a LLH executes, HHRL computes the resulting population Hypervolume, computed using the same reference points. The scheme counts how many times a LLH becomes the best one in all stages. These counts are then used to determine which LLH should compose the allowed set A. LLHs with performance worse than the average is not allowed to compose A. The algorithm continues by selecting a current LLH h_i according to the $\varepsilon-$RoulleteGreedy and returning all the generated information.

Algorithm 11: HHLA and HHRL Initialization Pseudocode

```
1  Input:
2  Problem;
3  $H$: set of LLHs $\{h_1, ..., h_i, ..., h_n\}$;
4  begin
5      $Pop \leftarrow$ Generate Random Population;
6      $P \leftarrow$ Create transition Matrix;
7      if HHLA then
8          $A \leftarrow H$;
9      end
10     else if HHRL then
11         $A \leftarrow SelectAllowedLLH(H)$;
12     end
13     $h_i \leftarrow$ Select first LLH to run;
14     return $A, P, Pop, h_i$
15 end
```

As mentioned, HHLA and HHRL can be classified as a Reinforcement Learning based online selection hyper-heuristic. For performance evaluation, the authors employed IBEA, SPEA2, and NSGAII as LLHs to find solutions for the WFG and DTLZ benchmarks and variants of the Crashworthiness problem. The results showed that this approach outperformed HHCF, making it one of the state-of-art online selection hyper-heuristic for multi-objective optimization.

Since this thesis is proposing a hyper-heuristic to deal with multi-objective optimization, HHCF, HHLA and especially HHRL will be considered in the empirical performance evaluation of the work proposed in this thesis.

6.3 Collaborative meta-heuristics designed as MAS

Some methods are not considered hyper-heuristics but proposed an interesting way to design collaborative meta-heuristics using multi-agent systems concepts. Talukdar et al. (TALUKDAR et al., 1998) propose A-Teams, a synergistic team of problem-solving methods which cooperates by sharing a population of candidate solutions. In this approach, there is no coordination or planning mechanism, and solutions are shared, through the central memory mechanism, allowing other agents to use these solutions in order to guide the search through promising search space, thus reducing the chances of being stuck at a local optimum.

Rabak and Sichman (RABAK; SICHMAN, 2003) extended A-Teams to design OPTIMA, an approach designed to solve the automatic electronic component insertion process on a particular inserting machine, named the Panasert AVK machine. Aydin and Fogarty (AYDIN; FOGARTY, 2004) extended the A-Teams approach for solving Job Shop Scheduling. They employed as problem-solving agents: SA (Simulated Annealing), TS (Taboo Search), HC (Hill Climbing), CSA (Simulated Annealing), CTS (Taboo Search), CHC (Hill Climbing), CHC2 (Hill Climbing), GA (Genetic Algorithm), NT (Improved version of CTS), and Damage. Barbucha (BARBUCHA, 2014) also extended the A-Team approach in order to create an Agent-Based Cooperative Population Learning Algorithm for the Vehicle Routing Problem with Time Windows. In his approach, the search is treated into stages, and different search procedures are used at each stage. The first stage is organized as an A-Team, where agents are used for improving the individuals stored in the collective memory. In the second stage, the individuals in the population (collective memory) are divided into subpopulations and allocated to a different set of A-Teams. At this level, each A-Team uses the same heuristics working under the same cooperation scheme. In the third stage, the sub-populations and the team of A-Teams architecture are also being employed. However, the process of communication among the set of A-Teams is used. The author evaluated his approach setting five problem-specific heuristics in the first stage and a set of four Tabu Search and simulated annealing in the higher level.

Milano and Roli (MILANO; ROLI, 2004) presented the Multi-agent Meta-heuristic Architecture (MAGMA), a four-level architecture, with one or more agents at each level, where each level one or more agents act. The first level contains solution builders agents, responsible for providing feasible solutions for upper levels. The second level contains solution improvers, responsible for providing local search and solution improvements until a termination condition is verified. The third-level agents have a global view of the search space, or, at least, their task is to guide the search towards promising regions, trying to avoid local optima. In the last level (Level-3), higher-level strategies are described, such as a cooperative search and any other combination of meta-heuristics. The authors showed that the three first levels are enough to describe standalone meta-heuristics and then evolutionary algorithms. Besides that, Level-3 can model coordinated and cooperative hybrid meta-heuristics.

Talbi and Bachelet (TALBI; BACHELET, 2006) propose a hybrid approach to solve the quadratic assignment problem by applying Tabu Search, Genetic Algorithm e KO (kick operator) as cooperative agents. The three heuristic agents run simultaneously and exchange information via an adaptive memory (AM). Each algorithm has a role: Tabu Search is used as the primary search algorithm, Genetic Algorithm is in charge of the diversification and Kick Operator is applied to intensify the search.

6.4 Discussion

Table 9 presents all cited work, by classifying them according to the following criteria:

- MAS: which approaches are designed as a multi-agent system;
- HH: which are considered a hyper-heuristic;
- Group 1: which papers treat heuristics as low-level-heuristics;
- Group 2: which treat meta-heuristics as low-level-heuristics;
- MOP: to identify which approach deals with multi-objective optimization;
- Cooperative (Coop.) or Competitive (Comp.): if LLHs try to cooperate with each other or compete for resources.

We can draw the following conclusions by analyzing Table 9:

- Most of the work here listed are considered hyper-heuristic;
- Most of the hyper-heuristic are not agent-based;

Table 9: Papers classification

Paper	MAS	HH	Group 1	Group 2	MOP	Coop.	Comp.
(TALUKDAR et al., 1998)	✓		✓	✓		✓	
(RABAK; SICHMAN, 2003)	✓		✓	✓		✓	
(AYDIN; FOGARTY, 2004)	✓		✓	✓		✓	
(BARBUCHA, 2014)	✓		✓	✓		✓	
(MILANO; ROLI, 2004)	✓		✓	✓		✓	
(TALBI; BACHELET, 2006)	✓		✓	✓		✓	
(CADENAS; GARRIDO; MUNOZ, 2007)	✓	✓		✓		✓	
(MALEK, 2010)	✓	✓		✓		✓	
(OUELHADJ; PETROVIC, 2010)	✓	✓	✓			✓	✓
(MEIGNAN; KOUKAM; CRÉPUT, 2010)	✓	✓	✓			✓	
(MARTIN et al., 2016)	✓	✓	✓	✓		✓	
(ACAN; LOTFI, 2016)	✓	✓		✓	✓	✓	
(NUGRAHENI; ABEDNEGO, 2016)	✓	✓		✓		✓	
(MCCLYMONT et al., 2013)		✓	✓		✓		✓
(KHEIRI; KEEDWELL, 2017)		✓	✓		✓		✓
(GUIZZO et al., 2015)		✓	✓		✓		✓
(GUIZZO; VERGILIO; POZO, 2015)		✓	✓		✓		✓
(CASTRO; POZO, 2015)		✓	✓		✓		✓
(CARVALHO, 2015)		✓	✓		✓		✓
(GONÇALVES et al., 2015)		✓	✓		✓		✓
(GONÇALVES; ALMEIDA; POZO, 2015)		✓	✓		✓		✓
(LI, K. et al, 2014)		✓	✓		✓		✓
(ALMEIDA et al., 2020)		✓	✓		✓		✓
(SABAR et al., 2015)		✓	✓				✓
(VÁZQUEZ-RODRÍGUEZ; PETROVIC, 2012)		✓		✓	✓	✓	✓
(VRUGT; ROBINSON, 2007)		✓		✓	✓	✓	✓
(MAASHI; ÖZCAN; KENDALL, 2014)		✓		✓	✓		✓
(CARVALHO; VERGILIO; POZO, 2015)		✓		✓	✓		✓
(Li; ÖZCAN; John, 2019)		✓		✓	✓		✓
THIS THESIS	✓	✓		✓	✓	✓	✓

- Most of the work belong to group 1 (use heuristics as LLH);
- Half studies deal with MOP;
- Most of the studies adopt either competitive or cooperative strategies. The minority of them consider both strategies at the same time.

The work presented in this thesis is the only one that concurrently (i) is classified as a HH, (ii) treats meta-heuristics as low-level-heuristics, (iii) finds solutions to MOPs, (iv) considers both competitive and cooperative strategies and (v) is designed as a MAS.

By analyzing Table 9 one can see three work very related work: (ACAN; LOTFI, 2016), (MAASHI; ÖZCAN; KENDALL, 2014) (HHCF) and (Li; ÖZCAN; John, 2019) (HHRL and HHLA). All of them consider MOEAs to be online selected while a multi-objective problem is solved. The two last, as performed in this thesis, do not use multiple populations and focus on how well the hyper-heuristic perform. In the first one, the authors employed multiple populations, and by doing that, giving a way more chances for solutions to survive. Moreover, HHCF was successfully applied in several domains. HHRL and HHLA overcome HHCF results, which makes this approach very interesting for comparisons.

6.5 Conclusion

This chapter presented a succinct literature review on hyper-heuristics for selecting heuristics/meta-heuristics. This review considered both agent-based and non-agent-based approaches. State-of-art hyper-heuristics for selecting MOEAs for solving MOPs, such as HHCF and HHLA/HHRL, were introduced. These HH are employed in empirical experiments presented in the next chapter.

PART III

EMPIRICAL EVALUATION

7 EXPERIMENTS

This chapter presents three experimental studies. In the first one (Section 7.2), MOABHH is instantiated using three different voting methods described in Chapter 4 in order to solve the benchmark WFG for two and three objectives. The second one (Section 7.3) also instantiates MOABHH using the three voting methods in order to find solutions for some real-world problems. In both Experiment I and Experiment II, the MOABHH performance was compared against the meta-heuristics outcome, which means, full instances of the meta-heuristics which are also employed as LLH. Finally, In Section 7.4 the biggest comparison is performed by considering 18 real-world problems, the three MOABHH instances (for each voting method) and three other state-of-art hyper-heuristic.

The results of the first, the second and the third experiments were originally published respectively in (CARVALHO; SICHMAN, 2017), (CARVALHO; SICHMAN, 2018b; CARVALHO et al., 2020) and (CARVALHO; ÖZCAN; SICHMAN, 2021).

7.1 Experimental Setup

All experiments were set according to literature recommendation. Table 10 presents parameters used for evolutionary operators. Parameters values from (DEB; JAIN, 2014; MAASHI; ÖZCAN; KENDALL, 2014; Li; ÖZCAN; John, 2019) were considered for *SBX Crossover* and *Polynomial Mutation* (for continuous optimization). For this mutation, the dv in the function $1/dv$ means the number of decision variables in the problem. For discrete optimization, *PMX Crossover* and *Permutation Swap Mutation*, parameters from (PSYCHAS; DELIMPASI; MARINAKIS, 2015) were considered. For *Differential Evolution* we considered parameters from (KUKKONEN; LAMPINEN, 2005).

Table 11 present algorithm parameters for all the three performed studies. Each algorithm (hyper-heuristic and MOEA) had 30 independent runs for each problem instance. We followed (BRADSTREET et al., 2007) for Experiment 1 regarding the number of generations and population size. We followed (MAASHI; ÖZCAN; KENDALL, 2014) for Experiment II and IIIA. In Experiment IIIB, we considered a smaller population of solutions and fewer generations to run for problems with very long execution (up to two days of execution for each

Table 10: Heuristics Parameters

Heuristic	distribution			rate
SBX Crossover	20			0.9
Polynomial Mutation	20			1/dv
PMX Crossover	-			0.9
Permutation Swap Mutation	-			0.2
	F	Cr	K	Type
Differential Evolution	0.2	0.2	0.5	rand/1/bin

run). In Experiment IIIA, HHCF parameters followed (MAASHI; ÖZCAN; KENDALL, 2014) while HHRL/HHLA considered (Li; ÖZCAN; John, 2019). For Experiment IIIB, these parameters were scalarized due to the different populations and generations demanded by the studied problems. Different parameters to MOABHH were set in different experiments based on tuning.

Table 11: Hyper-heuristics and MOEAs Parameters

	Experiment I	Experiment II	Experiment IIIA	Experiment IIIB
Number of independent runs	30	30	30	30
Number of generations	750	1000	1000	50
Population Size	100	100	100	30
MOABHH δ	50	50	50	1
MOABHH ϵ	50	50	50	1
MOABHH β	0.3	0.3	1	0.5
HHCF generations per round	-	-	25	1
HHCF α	-	-	30	30
HHRL/HHLA Decision Points	-	-	25	25
HHRL/HHLA m	-	-	2	2
HHRL/HHLA α	-	-	0.1	0.1
HHRL/HHLA β	-	-	3	3

The Hypervolume average was computed after every execution. For benchmark functions, true Pareto fronts (PF_t) provided by (HUBAND et al., 2006) were considered. For all the other problems, since they are real-world problems, there is no previous known optimal solution set. Thus, Pareto Known Fronts (PF_k) were created, for each problem instanced, by combining all the results got by all algorithms along with the 30 runs. Reference points for the Hypervolume indicator were taken for the available Pareto Front.

After obtaining all the averages, we compared the results using Kruskal-Wallis Statistical Test with a confidence level of 95%. So, in all hypervolume result tables, red values mean lower Hypervolume values with a statistical difference, black values mean averages statistically tied and **bold values** highlight those with the higher Hypervolume values, although not necessarily with a statistical difference.

For each experiment, in order to perform a cross-domain evaluation of algorithms, we followed (DERRAC et al., 2011) and generated the average Friedman ranking considering Hypervolume averages. This ranking considers which position each algorithm takes on each problem, and assign an average ranking for each algorithm.

In this thesis, a MOABHH instance occurs when this model is instantiated considering a voting method. For example, using voting methods detailed in Chapter 4. Thus, $MOABHH_C$ refers to MOABHH instantiated considering Copeland Voting, $MOABHH_B$ instantiated considering Borda and $MOABHH_K$ with Kemeny-Young.

Regarding the implementation, all MOEAs used are implemented by jMetal 5.7 (NEBRO; DURILLO; VERGNE, 2015). HHCF, MOABHH and HHLA/HHRL were implemented in Java JDK10.

7.2 Experiment I: Benchmark

7.2.1 Description

In this first experiment, the Walking Fish Group (Section 2.5.1) benchmark was employed in order to evaluate the performance of the proposed hyper-heuristic. This is a similar experiment like the one performed in (CARVALHO; SICHMAN, 2017). The difference lies in the fact that MOABHH is instanced here with three different voting methods: Borda, Copeland and Kemeny-Young, while in (CARVALHO; SICHMAN, 2017) only Copeland was used. These instances will be named along with this chapter as $MOABHH_B$ for Borda, $MOABHH_C$ for Copeland and $MOABHH_K$ for Kemeny.

This experiment setup followed (BRADSTREET et al., 2007; CARVALHO; SICHMAN, 2017) by setting $no \in \{2, 3\}$ as the number of objectives, $l = 20$ as the number of distance variables, and $k = 2 * (no - 1)$ as the number of positions (as stated in Section 2.5.1). Thus this experiment considers 18 benchmark problems to be optimized.

In this experiment, the MOEAs: NSGA-II, SPEA2 and IBEA were employed both as single instance algorithms and as LLHs for MOABHH. So, MOABHH has three candidates (LLH Agents) in the election. All MOABHH instances ($MOABHH_B$, $MOABHH_C$ and $MOABHH_K$) employed Hypervolume, HR, RNI, AE, UD and ER as quality indicators voters.

7.2.2 Results

Table 14 presents results for this experiment. IBEA is clearly the best algorithm here, followed by the three hyper-heuristics. For 2-objective problems, all three MOABHH instances found competitive results, especially $MOABHH_B$, by getting the higher hypervolume average on WFG2 and $MOABHH_C$ in WFG5. There is no statistical difference among the hypervolume averages for $MOABHH_B$, $MOABHH_C$, and $MOABHH_K$.

For 3-objective problems, all MOABHH did not find competitive results in WFG8. $MOABHH_C$ also was overcome in WFG4, WFG5 and WFG7-WFG9, and $MOABHH_K$ in WFG4, WFG7-WFG9. SPEA2 just find tied averages on WFG5, and NSGA-II found tied results in WFG2 and WFG5.

Table 12 summarizes the performance of the algorithms considering how many times they got the best averages, tied results and got worse results compared to the best one. IBEA performs well in all the problems. The three MOABHH hyper-heuristics, especially $MOABHH_B$, reached good results in most of the problems.

Table 12: Experiment I Summary

	IBEA	NSGAII	SPEA2	$MOABHH_B$	$MOABHH_C$	$MOABHH_K$
Best Average	13	2	0	1	1	1
Tied Average	5	3	3	16	12	13
Worse Average	0	13	15	1	5	4
Total Competitive (Best + Tied)	18	5	3	17	13	14

Using one algorithm much better than others (IBEA) is an interesting experiment because HH must find the best one as soon as possible. Using NSGA-II and SPEA2 more than necessary deteriorate performance in this experiment. $MOABHH_B$ was the best HH in this case because it chose more IBEA than the others.

Table 13 presents the cross-domain statistical evaluation performed by Friedman average ranking. In this table, smaller statistical values are considered as better. Here, IBEA has the best ranking, followed by $MOABHH_B$, Kemeny and $MOABHH_C$. NSGAII and SPEA2 are the two worst algorithms according to this ranking.

Table 13: Average Rankings of the algorithms (Friedman) for Experiment I

Algorithm	Ranking
IBEA	1.7777777777777781
NSGAII	4.3888888888888875
SPEA2	5.611111111111111
$MOABHH_B$	2.666666666666667
$MOABHH_C$	3.3333333333333335
$MOABHH_K$	3.2222222222222222

Table 14: Hypervolume Averages for WFG Experiment I

Obj.	Problem	IBEA	NSGAII	SPEA2	$MOABHH_B$	$MOABHH_C$	$MOABHH_K$
	WFG1	**7.4273E-01**	6.8839E-01	6.7419E-01	7.2754E-01	7.2524E-01	7.3761E-01
	WFG2	7.2123E-01	7.3058E-01	7.2775E-01	**7.3351E-01**	7.2806E-01	7.2815E-01
	WFG3	**7.2950E-01**	7.2695E-01	7.2315E-01	7.2931E-01	7.2949E-01	7.2934E-01
	WFG4	**5.7828E-01**	5.7547E-01	5.7240E-01	5.7819E-01	5.7815E-01	5.7818E-01
2	WFG5	5.4856E-01	5.4540E-01	5.4078E-01	5.4798E-01	**5.4957E-01**	5.4832E-01
	WFG6	**5.5696E-01**	5.5653E-01	5.5114E-01	5.5711E-01	5.5655E-01	5.5697E-01
	WFG7	**5.7858E-01**	5.7701E-01	5.7301E-01	5.7855E-01	5.7855E-01	5.7855E-01
	WFG8	**5.3721E-01**	5.3552E-01	5.3131E-01	5.3639E-01	5.3654E-01	5.3640E-01
	WFG9	5.4127E-01	**5.4849E-01**	5.4162E-01	5.4410E-01	5.3559E-01	5.3442E-01
	WFG1	**7.7216E-01**	5.2515E-01	4.5775E-01	7.4722E-01	7.5231E-01	7.4043E-01
	WFG2	8.8869E-01	**8.9224E-01**	8.8447E-01	8.8228E-01	8.7876E-01	8.7586E-01
	WFG3	7.1962E-01	7.1062E-01	6.6755E-01	7.1960E-01	7.1964E-01	**7.2036E-01**
	WFG4	**7.3278E-01**	6.7707E-01	6.3579E-01	7.3044E-01	7.1852E-01	7.1473E-01
3	WFG5	**7.0117E-01**	6.6143E-01	6.2448E-01	6.9777E-01	6.7579E-01	6.8487E-01
	WFG6	**7.0754E-01**	6.6282E-01	6.2105E-01	7.0462E-01	6.9569E-01	7.0517E-01
	WFG7	**7.3537E-01**	6.9242E-01	6.3869E-01	7.3290E-01	7.1595E-01	7.2235E-01
	WFG8	**6.7701E-01**	6.2164E-01	5.6177E-01	6.7505E-01	6.7017E-01	6.6649E-01
	WFG9	**6.8146E-01**	6.3760E-01	6.2128E-01	6.6282E-01	6.4573E-01	6.4880E-01

7.3 Experiment II: Discrete and continuous evaluation

7.3.1 Description

The second experiment aims to evaluate the MOABHH ($MOABHH_B$, $MOABHH_C$ and $MOABHH_K$) on solving real-world problems. This experiment was partially published in (CARVALHO; SICHMAN, 2018b; CARVALHO et al., 2020) and it is here presented with more complete results. For this purpose, four continuous optimization problems were considered: Car Side Impact, Crashworthiness, Water and Machining (Section 2.5.2). This experiment also considers the discrete optimization problem moTSP using 100, 300 and 500 cities evaluating instances with $2, 3, 4, 5$ objectives.

7.3.2 Results

For evaluation purposes, we compared MOABHH's result with each individual *MOEA agent*. Table 15 shows hypervolume averages in continuous engineering real-world problems. In this table, as well as in Table 16, black values mean statistically tied higher hypervolume averages, not having a statistical difference from the highest one, represented by a **bold value**. Red values are statistically overcome. $MOABHH_B$ found the best hypervolume average in CrashWorthiness (overcoming all MOEAs, but tied with our other HHs) and Water (overcoming other HHs). For Machining, $MOABHH_K$ got the best average tied with the best MOEA (IBEA) and the other HHs. However, in the Car Side Impact problem, it was overcome by IBEA (the best average), $MOABHH_C$ and $MOABHH_K$.

Table 15: Hypervolume results for continuous engineering real-world problems

Problem	GDE3	IBEA	NSGAII	SPEA2	$MOABHH_B$	$MOABHH_C$	$MOABHH_K$
Water	5.6966E-01	5.0861E-01	5.2592E-01	5.0309E-01	**5.8639E-01**	5.5992E-01	5.6089E-01
Machining	1.8626E-01	2.8005E-01	1.8670E-01	1.7927E-01	2.7674E-01	2.7444E-01	**2.8194E-01**
Car Side Impact	4.3793E-01	**4.7013E-01**	4.2184E-01	4.3920E-01	4.5609E-01	4.6116E-01	4.6511E-01
Crash worthiness	7.3630E-01	7.0617E-01	7.2944E-01	7.2236E-01	**7.3937E-01**	7.3859E-01	7.3814E-01

For moTSP, problems instances were instantiated with 100, 300 and 500 cities from DIMACS [1], a TSP benchmark composed by a set of 1-objective problems that can be combined to generate moTSP problems. Thus, for each problem, we combined the five 1-objective problems (A, B, C, D, and E), following Paquete *et al.* (PAQUETE; CHIARANDINI; STÜTZLE,

[1]https://eden.dei.uc.pt/ paquete/tsp/

2004). For example, a problem named *A-C-D* is a tri-objective optimization problem generated using the 1-objective problems A, C and D. In total, we employed 72 different discrete optimization problems.

Tables 16, 17 and 18 shows respectively hypervolume averages for 100, 300 and 500 cites problems. For 100 cities, no statistical difference was found between MOABHH instances and IBEA, the best algorithm found. NSGA-II and SPEA2 were outperformed in all problems with more than three objectives, sometimes finding just dominated solutions (when compared to other MOEA solutions), and so having zero as hypervolume averages. SPEA2 is the worse algorithm here, by just finding tied results in three problems (A-D, B-D, and D-E). NSGA-II just found tied results in 2-objectives problems.

In 300 cities experiments, $MOABHH_B$ and $MOABHH_C$ found tied results with IBEA in all problems. $MOABHH_K$ was outperformed in two problems (A-C and A-B-C-D-E). NSGA-II and SPEA2 were outperformed in all problems, sometimes with zero-valued hypervolumes averages.

In 500 cities experiments, $MOABHH_B$ was outperformed just in one problem (B-D). $MOABHH_C$ did not find tied results in 4 problems (A-B, B-E, A-B-C-D, and A-B-C-D-E) and $MOABHH_K$ in two problems (B-D and A-B-C-D). NSGA-II and SPEA2 were outperformed in all problems, sometimes with zero-valued hypervolumes averages.

Table 16: moTSP Hypervolume results for 100 cities problems

Obj.	Problem	IBEA	SPEA2	NSGA-II	$MOABHH_B$	$MOABHH_C$	$MOABHH_K$
2	A-B	**5.4035E-01**	4.7779E-01	4.9911E-01	5.0164E-01	5.0819E-01	5.1816E-01
	A-C	5.9319E-01	5.8360E-01	5.9474E-01	6.1971E-01	6.2208E-01	**6.2527E-01**
	A-D	4.5715E-01	3.9650E-01	**4.6830E-01**	4.4563E-01	4.4353E-01	4.2217E-01
	A-E	5.3212E-01	4.5169E-01	4.7707E-01	**5.2613E-01**	4.7766E-01	4.9609E-01
	B-C	4.6332E-01	4.3609E-01	4.8509E-01	**4.9080E-01**	4.6960E-01	4.7095E-01
	B-D	5.4431E-01	4.9306E-01	5.3926E-01	5.4578E-01	5.3647E-01	**5.5180E-01**
	B-E	5.1461E-01	4.6549E-01	5.0102E-01	5.0425E-01	5.1585E-01	**5.1999E-01**
	D-E	**5.1809E-01**	4.9690E-01	4.9600E-01	4.9925E-01	5.0892E-01	5.1148E-01
3	A-B-C	**4.8495E-01**	4.1456E-01	2.1440E-01	4.8220E-01	4.7829E-01	4.7885E-01
	A-B-D	5.1534E-01	4.4888E-01	2.6171E-01	**5.2126E-01**	5.0962E-01	5.0569E-01
	A-B-E	**4.6178E-01**	3.5999E-01	1.8964E-01	4.5840E-01	4.5377E-01	4.5109E-01
	A-C-D	5.6023E-01	4.9321E-01	3.1264E-01	**5.6637E-01**	5.5246E-01	5.4835E-01
	A-C-E	5.1899E-01	4.4108E-01	2.6199E-01	5.2616E-01	5.1975E-01	**5.3756E-01**
	A-D-E	**5.0983E-01**	3.9365E-01	2.0880E-01	4.8518E-01	5.0332E-01	4.8770E-01
	B-C-D	**4.7709E-01**	3.7966E-01	2.5222E-01	4.4362E-01	4.5226E-01	4.6201E-01
	B-C-E	4.9138E-01	4.0281E-01	2.0546E-01	4.8864E-01	**4.9444E-01**	4.8290E-01
	B-D-E	3.8455E-01	2.8662E-01	1.2573E-01	3.8934E-01	**3.9322E-01**	3.7559E-01
	C-D-E	4.9682E-01	4.1544E-01	2.5733E-01	4.9340E-01	4.8724E-01	**5.0100E-01**
4	A-B-C-D	2.5594E-01	6.7345E-02	0.0000E+00	**2.6236E-01**	2.4459E-01	2.6282E-01
	A-B-C-E	**4.2801E-01**	2.3812E-01	1.8766E-03	4.1034E-01	4.0248E-01	4.0576E-01
	A-B-D-E	2.6658E-01	7.3478E-02	0.0000E+00	2.4563E-01	**2.6982E-01**	2.6752E-01
	A-C-D-E	**4.2391E-01**	2.5905E-01	3.2653E-03	3.9823E-01	3.9784E-01	4.1429E-01
	B-C-D-E	2.5577E-01	5.5816E-02	0.0000E+00	2.4217E-01	**2.6454E-01**	2.4721E-01
5	A-B-C-D-E	**1.8162E-01**	7.6230E-04	0.0000E+00	1.7742E-01	1.7585E-01	1.7188E-01

Table 17: moTSP Hypervolume results for 300 cities problems

Obj.	Problem	IBEA	SPEA2	NSGA-II	$MOABHH_B$	$MOABHH_C$	$MOABHH_K$
2	A-B	**3.4477E-01**	1.0628E-01	1.9837E-01	3.4556E-01	2.5192E-01	2.9158E-01
	A-C	**1.9181E-01**	6.9629E-03	3.9852E-02	9.7699E-02	1.2922E-01	1.0086E-01
	A-D	**2.1098E-01**	1.4334E-02	7.9636E-02	2.0182E-01	1.3524E-01	1.4742E-01
	A-E	**2.9146E-01**	5.7068E-02	1.4574E-01	2.4915E-01	2.4967E-01	2.4736E-01
	B-C	**2.8195E-01**	3.9052E-02	8.1285E-02	1.6298E-01	2.1979E-01	1.7911E-01
	B-D	**8.9690E-02**	3.1471E-03	1.1332E-02	6.1996E-02	4.4891E-02	5.2866E-02
	B-E	**2.7389E-01**	3.9910E-02	9.5145E-02	2.0957E-01	1.7410E-01	2.0914E-01
	D-E	**1.7335E-01**	3.2888E-03	2.1784E-02	8.9845E-02	1.1050E-01	9.7053E-02
3	A-B-C	**2.4760E-01**	1.0342E-02	0.0000E+00	2.1426E-01	2.3443E-01	2.3034E-01
	A-B-D	**2.0451E-01**	5.9115E-03	0.0000E+00	1.8212E-01	1.9698E-01	1.7616E-01
	A-B-E	**2.7147E-01**	1.9147E-02	0.0000E+00	2.6446E-01	2.5941E-01	2.5173E-01
	A-C-D	**2.6918E-01**	2.2637E-02	3.6814E-05	2.5458E-01	2.3914E-01	2.5375E-01
	A-C-E	**2.5521E-01**	3.8172E-03	0.0000E+00	2.2481E-01	2.1414E-01	2.2447E-01
	A-D-E	**2.3968E-01**	1.7239E-02	3.9118E-06	2.1167E-01	2.0448E-01	1.9552E-01
	B-C-D	**2.7007E-01**	1.8408E-02	0.0000E+00	2.3418E-01	2.5367E-01	2.6510E-01
	B-C-E	**3.3973E-01**	4.5529E-02	5.8039E-05	2.9439E-01	2.7783E-01	2.9915E-01
	B-D-E	1.9336E-01	1.9119E-03	0.0000E+00	1.5801E-01	1.7727E-01	**1.9765E-01**
	C-D-E	**2.2654E-01**	6.1922E-03	0.0000E+00	2.0187E-01	2.2495E-01	2.1348E-01
4	A-B-C-D	**1.5264E-01**	1.7043E-08	0.0000E+00	1.2746E-01	1.2331E-01	1.2074E-01
	A-B-C-E	**1.4398E-01**	1.0926E-05	0.0000E+00	1.2276E-01	1.1770E-01	1.1338E-01
	A-B-D-E	**1.6575E-01**	1.9504E-04	0.0000E+00	1.2810E-01	1.3119E-01	1.3793E-01
	A-C-D-E	1.8510E-01	7.0819E-06	0.0000E+00	**1.8532E-01**	1.6692E-01	1.4873E-01
	B-C-D-E	**1.4505E-01**	0.0000E+00	0.0000E+00	1.2333E-01	1.0074E-01	1.0481E-01
5	A-B-C-D-E	**1.0370E-01**	0.0000E+00	0.0000E+00	7.0018E-02	7.5318E-02	6.5663E-02

Table 18: moTSP Hypervolume results for 500 cities problems

Obj.	Problem	IBEA	SPEA2	NSGA-II	$MOABHH_B$	$MOABHH_C$	$MOABHH_K$
2	A-B	**3.0971E-01**	2.7333E-02	1.2989E-01	2.2817E-01	2.2056E-01	2.5826E-01
	A-C	**3.9296E-01**	4.3744E-02	1.3494E-01	2.8918E-01	2.7929E-01	2.9652E-01
	A-D	**3.2423E-01**	3.2357E-02	1.0020E-01	2.4799E-01	2.5170E-01	2.5032E-01
	A-E	2.9747E-01	2.6163E-02	1.1451E-01	2.0568E-01	2.3650E-01	2.1522E-01
	B-C	**2.6316E-01**	3.8700E-03	4.2782E-02	1.6620E-01	1.8824E-01	1.8270E-01
	B-D	**1.6155E-01**	5.6406E-03	2.9149E-02	8.5206E-02	9.2730E-02	7.5866E-02
	B-E	**2.1235E-01**	3.8428E-04	2.6925E-02	1.2112E-01	1.0454E-01	1.4513E-01
	D-E	**2.3617E-01**	1.1130E-02	4.9594E-02	1.8893E-01	1.5481E-01	1.8005E-01
3	A-B-C	**2.2600E-01**	3.0933E-03	0.0000E+00	1.8006E-01	1.7866E-01	1.9422E-01
	A-B-D	**2.0955E-01**	2.9666E-03	0.0000E+00	2.0363E-01	2.0868E-01	2.0177E-01
	A-B-E	**3.1399E-01**	4.4502E-03	0.0000E+00	2.3487E-01	2.5954E-01	2.5441E-01
	A-C-D	**2.4820E-01**	9.0777E-03	0.0000E+00	2.1905E-01	2.1984E-01	1.9709E-01
	A-C-E	**2.1200E-01**	2.6963E-03	0.0000E+00	1.5247E-01	1.4886E-01	1.4463E-01
	A-D-E	**2.3527E-01**	8.6451E-03	0.0000E+00	1.9279E-01	1.9355E-01	1.9894E-01
	B-C-D	**2.3875E-01**	3.8390E-03	0.0000E+00	1.8534E-01	1.7833E-01	1.8980E-01
	B-C-E	**1.8803E-01**	1.3207E-03	0.0000E+00	1.4518E-01	1.7380E-01	1.4003E-01
	B-D-E	**2.2777E-01**	3.9015E-03	0.0000E+00	2.1482E-01	2.0192E-01	2.0024E-01
	C-D-E	**1.8113E-01**	2.4474E-03	0.0000E+00	1.4559E-01	1.6310E-01	1.5891E-01
4	A-B-C-D	1.2815E-01	0.0000E+00	0.0000E+00	8.6029E-02	7.0892E-02	8.0518E-02
	A-B-C-E	**1.5159E-01**	0.0000E+00	0.0000E+00	1.0910E-01	1.1653E-01	1.0825E-01
	A-B-D-E	**1.5355E-01**	0.0000E+00	0.0000E+00	1.0443E-01	1.1226E-01	1.1089E-01
	A-C-D-E	**1.4316E-01**	0.0000E+00	0.0000E+00	1.0845E-01	1.0782E-01	1.1196E-01
	B-C-D-E	**1.7655E-01**	0.0000E+00	0.0000E+00	1.2994E-01	1.4604E-01	1.2650E-01
5	A-B-C-D-E	**1.0434E-01**	0.0000E+00	0.0000E+00	6.6333E-02	6.1503E-02	6.7028E-02

Table 19 summarizes the performance of the algorithms considering how many times they got the best averages, tied results and got worse results compared to the best one. Considering the continuous evaluation, no algorithm performed well in all of the problems. The three MOABHH hyper-heuristics were the best-studied algorithms with three good results, followed by IBEA. NSGAII and SPEA2 did not get good results. Considering moTSP, IBEA and $MOABHH_B$ performed well in almost all the problems. IBEA had been overcome in A-C bi-objective 100 cities problems while $MOABHH_B$ was overcome in B-D bi-objective 500 cities. In fact, $MOABHH_B$ is the best algorithm here (having good results in 74 of 76 problems), followed by IBEA (having good results in 73 of 76 problems). $MOABHH_C$ and $MOABHH_K$ tied, having good results in 71 of 76 problems.

Table 19: Experiment II Summary

		IBEA	NSGAII	SPEA2	GDE3	$MOABHH_B$	$MOABHH_C$	$MOABHH_K$
Continuous	Best average	1	0	0	0	2	0	0
	Tied average	1	0	0	1	1	3	3
	Worse average	2	4	4	3	1	1	1
moTSP 100	Best average	9	1	0	-	5	4	5
	Tied average	14	7	3	-	19	20	19
	Worse average	1	16	21	-	0	0	0
moTSP 300	Best average	22	0	0	-	1	0	1
	Tied average	2	0	0	-	23	24	21
	Worse average	0	24	24	-	0	0	2
moTSP 500	Best average	22	0	0	-	0	0	0
	Tied average	2	0	0	-	23	20	22
	Worse average	0	24	24	-	1	4	2
Total Competitive (Best + Tied)		73/76	8/76	9/76	1/4	74/76	71/76	71/76

Table 20 presents the statistical evaluation, similar to the one presented in Experiment I. For this purpose Tables 15, 16, 17 and 18 were considered. In order to summarize all results from experiment II, GDE3 was not considered in this ranking because it was not applied in moTSP experiments. Removing GDE3 from this analysis is not critical due to the fact of his bad performance in the four studied continuous problems.

In this ranking, IBEA also had the best ranking, followed by $MOABHH_B$, $MOABHH_K$ and $MOABHH_C$. NSGAII and SPEA2 are also the two worst algorithms according to this ranking. Interestingly, this strong and robust behavior of the Borda rule has been observed in other domains, such as when aggregating ranking information from crowd workers (MAO; PROCACCIA; CHEN, 2013).

Table 20: Average Rankings of the algorithms (Friedman) for Experiment II

Algorithm	Ranking
IBEA	1.5263157894736823
SPEA2	5.381578947368425
NSGAII	5.460526315789473
$MOABHH_B$	2.815789473684209
$MOABHH_C$	2.947368421052631
$MOABHH_K$	2.868421052631578

7.4 Experiment III: Real-world continuous problems

7.4.1 Description

In the third experiment, published in (CARVALHO; ÖZCAN; SICHMAN, 2021), the hyper-heuristics HHCF (Section 6.2.2.1), HHRL and HHLA (Section 6.2.2.2), and three MOABHH (Chapter 5) voting methods were eveluated. They controlled five LLH: GDE3, IBEA, NSGAII, SPEA2, and mIBEA (here added because its performance in (Li et al., 2017)). These MOEAs are used to solve the eighteen continuous real-world optimization problems presented in Table 21. These problems were selected due to the fact that there is no clear winner among the meta-heuristics. In this scenario, hyper-heuristics are necessary in order to find a good trade-off.

Different configurations were employed for these problems. Table 22 presents how many generations and population size was used for each problem. We set P17-P18 differently due to the high computational effort demanded in these applications, which takes almost three months for experiments using the same setup considered for the problems P01-P16.

7.4.2 Results

Table 23 presents Hypervolume averages: bold values are the best Hypervolume values among all nine algorithms, and black values are tied statistically with the best value.

From the experiments, we could conclude the following:

- if we compare the results obtained *just by the five MOEAs*, we could see that: (i) GDE3 obtained the highest results on five problems (P01, P04, P14, P16, and P17); (ii) IBEA was the best one on four problems (P02, P05, P07 and P13); (iii) NSGAII has the best

Table 21: A brief description real-world multi-objective problems containing the number of objectives, variables constraints and source

ID	Problem name	Objs.	Variables.	Constraints
P01	Water	5	3	7
P02	Machining	4	3	3
P03	CarSideImpact	3	7	0
P04	CrashWorthiness	3	5	0
P05	FourBarTruss	2	4	0
P06	Golinski	2	7	11
P07	Quagliarella	2	16	0
P08	Poloni	2	2	0
P09	Vibrating Platform Design	2	5	0
P10	Optical Filter	2	11	0
P11	Welded Beam Design	2	4	0
P12	Disk Brake Design	2	4	0
P13	Heat Exchanger	2	16	0
P14	Hydro Dynamics	2	6	0
P15	Area Under Curve	2	10	0
P16	Kernel Ridge Regression	2	5	0
P17	Facility Placement	2	20	0
P18	Neural Network Controller	2	24	0

Table 22: Parameters used in experiments

Experiment	Problems	Generations	Population Size
IIIA	P01-P16	1000	100
IIIB	P17-P18	50	30

average on four problems (P06, P10, P15, and P18); (iv) SPEA2 have the best average on five problems (P03, P08, P09, P11, and P12); and (v) mIBEA was one of the best in P05.

- if we consider *just the three MOABHH instances* we have: (i) $MOABHH_B$ has higher averages on 5 problems (P01, P07, P10, P13, P18); (ii) $MOABHH_C$ has higher averages on 9 problems (P02, P03, P04, P06, P09, P11, P12, P14 P17); (iii) $MOABHH_K$ has higher averages on 4 problems (P05, P08, P15, P16);

- if we consider *all the six hyper-heuristics,* we have: (i) $MOABHH_B$ has higher averages on four problems (P01, P07, P10, P13); (ii) $MOABHH_C$ has higher averages on four problems (P04, P06, P12, P17); (iii) $MOABHH_K$ has higher averages on three problems (P08, P15, P16); (iv) HHLA has its best results on two problems (P05, P18); (v) HHRL performs better than others in 4 problems (P02, P09, P11, P14); (vi) HHCF is the best algorithm just in P03.

- finally, if we compare the results obtained by *all 11 algorithms*, GDE3 have higher Hypervolume values on five problems (P01, P04, P14, P16 and P17), IBEA has better averages on two problems (P05 and P07), NSGAII has the best result on three problems (P06, P10, and P18), SPEA2 on three problems (P08, P09 and P11), mIBEA has it for P05, $MOABHH_B$ has a better average on P13, $MOABHH_C$ on P12, and $MOABHH_K$ on P14. HHRL has the best average in P02 and HHCF in P03. Moreover, HHLA did not excel in any problem.

Table 23: Experiment III Hypervolume averages

Problem	GDE3	IBEA	NSGAII	SPEA2	mIBEA	$MOABHH_B$	$MOABHH_C$	$MOABHH_K$	HHLA	HHRL	HHCF
P01	**2.84791E+24**	2.08488E+24	2.66287E+24	2.71790E+24	2.57745E+24	2.84604E+24	2.84442E+24	2.77045E+24	2.82384E+24	2.83082E+24	2.75174E+24
P02	1.20897E+01	1.25610E+01	1.17707E+01	1.25111E+01	1.25435E+01	1.21048E+01	1.22152E+01	1.20754E+01	1.25842E+01	**1.25956E+01**	1.22624E+01
P03	3.40420E+01	3.28792E+01	3.33962E+01	3.42734E+01	3.28450E+01	3.31287E+01	3.39539E+01	3.27408E+01	3.42207E+01	3.43544E+01	**3.44417E+01**
P04	**4.12492E+01**	3.91803E+01	4.10414E+01	4.10798E+01	3.91526E+01	4.10824E+01	4.12221E+01	4.09849E+01	4.09005E+01	4.11760E+01	4.09022E+01
P05	4.29229E+01	**4.31195E+01**	4.29357E+01	4.29909E+01	**4.31195E+01**	4.30417E+01	4.30361E+01	4.31114E+01	4.31168E+01	4.31036E+01	4.30303E+01
P06	1.83777E+06	1.83368E+06	**1.83818E+06**	1.83533E+06	1.83375E+06	1.83638E+06	1.83721E+06	1.83707E+06	1.83543E+06	1.83605E+06	1.83718E+06
P07	4.68216E+00	**4.73903E+00**	4.64485E+00	4.68849E+00	4.73045E+00	4.72856E+00	4.72592E+00	4.72810E+00	4.72761E+00	4.71728E+00	4.67220E+00
P08	3.68066E+02	3.63116E+02	3.68054E+02	**3.68172E+02**	3.65368E+02	3.68039E+02	3.68035E+02	3.68054E+02	3.65596E+02	3.67869E+02	3.66795E+02
P09	8.86096E-01	8.84300E-01	8.86688E-01	**8.86949E-01**	8.83161E-01	8.84177E-01	8.86158E-01	8.84403E-01	8.85940E-01	8.86519E-01	8.84936E-01
P10	7.64228E-01	7.56240E-01	**7.68136E-01**	7.67995E-01	7.52942E-01	7.68120E-01	7.67354E-01	7.66666E-01	7.67507E-01	7.67569E-01	7.62415E-01
P11	7.51232E-01	7.51447E-01	7.52182E-01	**7.53341E-01**	7.48007E-01	7.51090E-01	7.51937E-01	7.51220E-01	7.52904E-01	7.53102E-01	7.52211E-01
P12	7.39104E-01	7.30254E-01	7.39384E-01	7.40054E-01	7.30272E-01	7.35503E-01	**7.40063E-01**	7.38684E-01	7.36341E-01	7.38086E-01	7.38719E-01
P13	4.46843E-01	4.60968E-01	4.54099E-01	4.51260E-01	4.55125E-01	**4.61347E-01**	4.60597E-01	4.59792E-01	4.50648E-01	4.60969E-01	4.50661E-01
P14	**6.56092E-01**	6.52218E-01	6.54558E-01	6.54922E-01	6.45212E-01	6.55455E-01	6.55529E-01	6.55423E-01	6.54099E-01	6.55615E-01	6.51219E-01
P15	9.79673E-01	9.75097E-01	9.80961E-01	9.79463E-01	9.78678E-01	9.90872E-01	9.89425E-01	**9.91282E-01**	9.89091E-01	9.88244E-01	9.87321E-01
P16	**9.01918E-01**	8.36830E-01	7.28744E-01	6.46054E-01	5.61843E-01	8.66175E-01	8.47810E-01	8.71773E-01	7.87492E-01	7.08809E-01	6.90451E-01
P17	**9.07392E-01**	8.68258E-01	9.03361E-01	9.01765E-01	8.91543E-01	8.94496E-01	8.94115E-01	8.94496E-01	8.94128E-01	8.67591E-01	8.92694E-01
P18	5.28786E-01	5.04137E-01	**5.29036E-01**	4.97061E-01	5.00425E-01	4.94500E-01	4.93520E-01	4.91520E-01	4.96037E-01	3.76589E-01	4.78611E-01

Table 24 summarizes the performance of the algorithms considering how many times they got the best averages, tied results and got worse results compared to the best one.

No algorithm performed well in all of the problems. In fact, IBEA, NSGAII and mIBEA got worse averages than competitive ones. GDE3, $MOABHH_B$ and HHCF performed well in half of the problems. SPEA2, $MOABHH_K$ and HHLA solved well the simple majority of the problems (10 problems). HHRL and $MOABHH_C$ were the two best algorithms having 12 problems solved with good performance.

Table 24: Experiment III Summary

	GDE3	IBEA	NSGAII	SPEA2	mIBEA	$MOABHH_B$	$MOABHH_C$	$MOABHH_K$	HHLA	HHRL	HHCF
Best Average	5	2	3	3	1	1	1	1	0	1	1
Tied Average	4	5	3	7	5	8	11	9	10	11	8
Worse Average	9	11	12	8	12	9	6	8	8	6	9
Total Competitive (Best + Tied)	9	7	6	10	6	9	12	10	10	12	9

Table 25 presents the statistical evaluation, similar to those presented in Experiment I and II. According to this ranking, $MOABHH_C$ was the best-studied algorithm, followed by HHRL, GDE3 and $MOABHH_B$. mIBEA was the worst algorithm studied, while HHCF was the worst studied hyper-heuristics. $MOABHH_K$ was the worst hyper-heuristic among MOABHH instances.

Table 25: Average Rankings of the algorithms (Friedman) for Experiment III

Algorithm	Ranking
GDE3	4.944444444444443
IBEA	7.555555555555555
NSGAII	5.61111111111111
SPEA2	5.388888888888888
mIBEA	8.166666666666666
$MOABHH_B$	5.333333333333332
$MOABHH_C$	4.722222222222221
$MOABHH_K$	5.944444444444445
HHLA	6.055555555555556
HHRL	5.055555555555556
HHCF	7.222222222222223

7.4.3 Use of Low-level Meta-Heuristics

In this section, we address the following issue: how much a single LLH is chosen by a particular hyper-heuristics. Figure 34 graphically presents this usage: for MOABHH instances

($MOABHH_B$, $MOABHH_C$ and $MOABHH_K$), it represents the percentage of participation in generating offspring along with the search. On the other hand, for HHRL, HHLA, and HHCF, the figure presents the percentage of times that each LLH was chosen. The data considers all problem instances, each of them running 30 times.

The particular behavior of each HH may be found by analyzing Figure 34. For example, if we consider problem P03, one may notice that MOABHH instances have chosen more times GDE3 (blue) and SPEA2 (red), while HHLA chose more often SPEA2. On the other hand, HHRL and HHCF have chosen LLHs more uniformly. Table 26 presents a summarized evaluation of this analysis, where we classified HHs behavior in 4 classes: (i) *One Elitist*: problems where one LLH is clearly selected more times than any other[2]; (ii) *Two Elitist*: problems where two LLHs are priviledged[3]; (iii) *Three Elitist*: problems where three LLHs are selected more times than others[4]; (iv) *Not Elitist*: when there is no clear LLH preference.

Table 26: How elitist HHs are on selection LLHs

	One Elitist	Two Elitist	Three Elitist	Not Elitist
$MOABHH_B$	P01, P03, P05, P09, P11, P12, P13	P02, P04, P07, P08, P14	P10, P16	P06, P15, P17, P18
$MOABHH_C$	P01, P03, P04, P05, P09, P12	P02, P07, P08, P10, P11, P13, P14	P15, P16	P06, P17, P18
$MOABHH_K$	P01, P03, P06, P09, P11, P12, P16	P02, P04, P05, P07, P08, P10, P14, P15	P13	P17, P18
HHLA	P01, P03, P07, P08, P09, P11, P12, P13, P17	P04, P05, P10	P14, P16, P18	P02, P06, P15
HHCF	P01, P06, P11, P12, P16, P17, P18	P04, P07	P05, P08, P10, P15	P02, P03, P09, P13, P14
HHRL	P01, P08, P11, P12, P13	P04, P05, P06, P07, P09	P02, P03, P10, P14	P15, P16, P17, P18

We can identify HHLA and HHCF with more problems classified as *One Elitist*, while MOABHH and HHRL had more problems in *Two Elitist* category. For *Three Elitist*, HHLA,

[2]We have considered a threshold above 50% for this situation.

[3]We have considered a threshold above 40% for each LLH for this situation.

[4]We considered a threshold above 30% for each LLH for this situation.

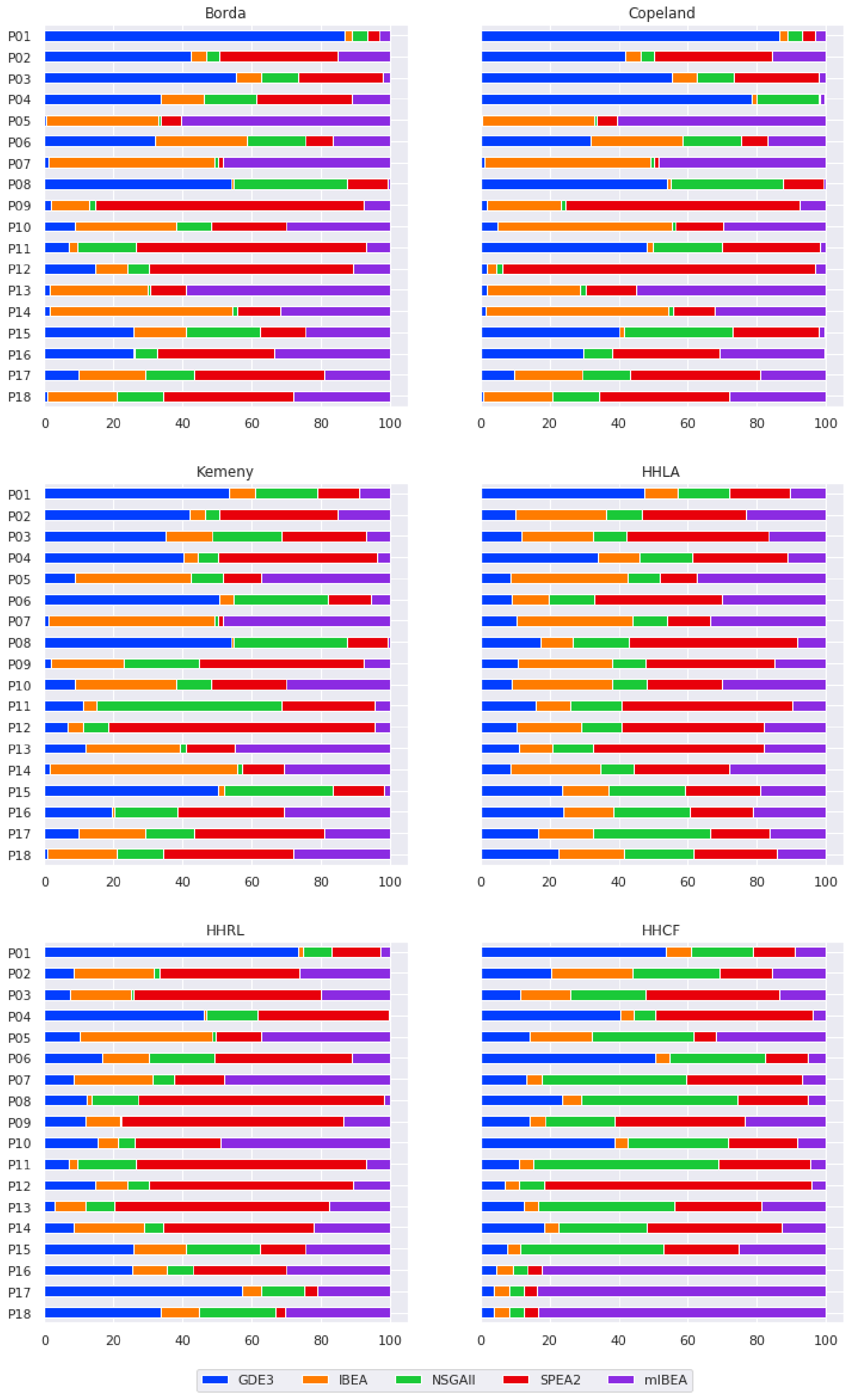


Figure 34: Utilization rate for the six hyper-heuristics

HHCF and HHRL had four problems classified while MOABHH had three. Considering all elitist classified problems (*One Elitist* + *Two Elitist* + *Three Elitist*) we can identify all HH behaving in a similar way.

7.5 Conclusion

This chapter presents three experiments. The first evaluated MOABHH instances on solving the WFG benchmark for two and three objectives. The second one evaluated MOABHH instances solving four real-world continuous problems and one discrete problem instantiated with different numbers of objectives. The third experiment compared MOABHH instances, state-of-art hyper-heuristics and MOEAs on solving eighteen different continuous real-world applications.

The first experiment showed $MOABHH_C$ employing more the best LLH, in this case, IBEA, and having competitive results against the best MOEA. $MOABHH_B$ was the best hyper-heuristic here. However, this is a benchmark experiment, which means this is limited for drawing a concrete conclusion on the performance of the algorithms. However, this kind of experiment was essential for the development of this research.

In the second experiment, instead of a benchmark, real-world problems were considered. In these problems, we also have a great performing algorithm, so hyper-heuristics have as soon as possible identify it and apply it in order to have competitive results against this best algorithm. IBEA is also the best MOEA here. Taking into account the number of times with competitive averages, that means, best hypervolume averages or averages statistically tied with the best, $MOABHH_B$ is the best algorithm. Considering the Friedman Ranking for this experiment, IBEA is also the best-studied algorithm.

Thus, this research reached the need for having a set of optimization problems where clearly there is no single best algorithm. This was the primary objective of experiment III. This made it the hardest test for the hyper-heuristic, but the most important to test them on real cross-domain applications. At this point in the research, there was the need of making MOABHH compete against state-of-art hyper-heuristics. As a result, HHRL and $MOABHH_C$ tied as the best algorithm according to the number of competitive results while $MOABHH_K$ and HHLA tied as the second-best. According to Friedman Ranking, $MOABHH_C$ was considered the best-studied algorithm, followed by HHRL and $MOABHH_B$.

Experiments I and II have different conclusions as to the one presented by Experiment III. The complexity of Copeland deteriorate results when there is a clear winner, but it helps when there is no single winner in all of the problems.

The following chapters present future research and additional information.

8 CONCLUSIONS AND FURTHER WORK

Over the years, multi-objective optimization has been tackled by meta-heuristics providing researchers and engineers with an interesting way to solve their problems. In the past decade, hyper-heuristics focused on multi-objective optimization have been designed with the objective to improve search results and diminish the effort on choosing meta-heuristic components or algorithms by themselves. This research belongs to this field, more precisely, it can be defined as an online selection hyper-heuristic focused on selecting the most suitable algorithm, more specifically a multi-objective evolutionary algorithm. Recent research has focused on this kind of hyper-heuristic, and this includes this present research.

In this research, the Algorithm Selection Problem for multi-objective optimization was attacked by a new approach designed using knowledge from Multi-Agent Systems and Social Choice Theory. In this approach, algorithms were considered as candidates in one election and quality indicators as voters.

This was not the first time researchers employed voting methods to deal with the ASP. Despite that, this was the first time voting methods were employed for designing hyper-heuristics.

As a multi-agent system, all components of this proposed hyper-heuristic could run in parallel, allowing evolutionary algorithms to evolve the same population, each one with his share. Thus, performance evaluations were more accurate because all those algorithms run at the same search moment making elections fairer. This is a concept which differs this research from other state-of-art hyper-heuristics that do not implement the concept of parallelism.

The first studies focused on how the proposed hyper-heuristic was compared to a clear best algorithm on solving benchmark functions. This was really necessary to verify whether the hypothesis could be plausible. After this, some real-world applications were introduced. Also, three different voting methods were considered. From this point, it was demonstrated how this approach could help researchers and engineers to diminish their effort in choosing multi-objective evolutionary algorithms. However, these studies were limited to one great performing algorithm to be identified. Thus, a study that clearly demonstrates the effect of the *No Free Lunch* theorem was needed. This is the context of a *Cross-Domain Application*, where hyper-heuristic research compare their algorithms on several scenarios in order to identify which hyper-heuristic performs better. Thus, state-of-art hyper-heuristics were considered and the proposed hyper-heuristic faced her hardest test. Results showed the proposed hyper-heuristic as very competitive against state-of-art hyper-heuristic. In fact, it overcomes them. This was also the first time this kind of hyper-heuristic was submitted to several real-world applications in the same study. Thus this effort was not just necessary to provide empirical results for this research, but also to deliver to the scientific community how this kind of algorithm behaves on real-world cross-domain applications.

From a practitioner's point of view, using the proposed approach can save a user from wasting a lot of time trying to run a tuning method to find the best metaheuristic to solve a new problem. If, on the one hand, the proposed approach was not the one that obtained the best result in all the problems studied in the experiments carried out, on the other hand, it was not the one that obtained the worst results, having a good performance in most of them. Indeed, MOABHH had the best performance considering the set of all problems from all experiments, precisely because there is no meta-heuristic that has the best performance in all of them, hence the interest of the adopted approach.

After performing all the three experiments, the research questions presented in this thesis introduction were answered as follows:

RQ1 How does the proposed hyper-heuristic's performance change when applied to different MOPs?
Several problems were considered ranging from benchmarks to real-world problems. No meta-heuristic was the best in all of the problems, especially on Experiment III, where one MOEA can be good for one problem and bad-performing for another one. In this scenario, we expect hyper-heuristics to have good performance. MOABHH did well, found good solutions (according to Hypervolume) in most of the studied problems.

RQ2 How does the hyper-heuristic performance change when using different voting methods?
Overall, Kemeny-Young was the worst voting method studied. Borda clearly gave MOABHH better performance on benchmarks (Experiment I), while Copeland helped MOABHH to better solve real-world problems from Experiment III. For Experiment II, both Copeland and Borda got really close results. It is possible to say that the complexity of Copeland deteriorates MOABHH performance when there is a clear winner, but it

helps when there is no single winner in all of the problems.

As future work, this hyper-heuristics will be studied across various applications in the discrete multi-objective optimization domain, such as Search-Based Software Engineering Problems (HARMAN; JONES, 2001). The idea behind it is finding a group of real-world problems where there is no clear best algorithm, the same as performed for continuous optimization, but for discrete optimization. Another possible direction is to test different meta-heuristics not classified as genetic algorithms such as MOEA/DD (LI et al., 2015b) (based on decomposition) and SMPSO (NEBRO et al., 2009) (based on swarm intelligence). Such meta-heuristics can produce interesting LLHs, since these have shown interesting solutions on solving continuous optimization problems. Voters can also employ different learning methods and then different perspectives from the same quality indicator can be obtained.

Regarding voting methods, their influence in the final result can be measured by running MOABHH considering just one voter, i.e. one quality indicator, so no voting method would be employed. If results deteriorate, then voting methods and multiple quality indicators are necessary. Other voting methods also can be evaluated in order to better study the influence of voting methods on MOABHH.

Considering multi-agent systems, some techniques like negotiation could be incorporated into the population sharing step. Since this step is currently performed randomly, using some intelligent agent models could generate interesting results.

Furthermore, recent hyper-heuristics such as (JÚNIOR; ÖZCAN; CARVALHO, 2020) and (FRITSCHE; POZO, 2019; FRITSCHE; POZO, 2020) can be compared against MOABHH.

9 LIST OF PUBLICATIONS

Portions of the content described in this thesis have been previously published in Conferences and Journal, as summarized in Table 27.

Table 27: Scientific production made during my PhD at USP/UoN

Title	Conference/ Journal	Observations
Applying Copeland Voting to Design an Agent-Based Hyper-Heuristic (CARVALHO; SICHMAN, 2017)	AAMAS'17	Most important conference on MAS. Preliminary MOABHH architecture. Experiment I
Multi-Agent Election-Based Hyper-Heuristics (CARVALHO; SICHMAN, 2018a)	IJCAI'18	Doctoral Consortium MOABHH architecture
Solving real-world multi-objective engineering optimization problems with an Election-Based Hyper-Heuristic (CARVALHO; SICHMAN, 2018b)	OptMAS'18	IJCAI Workshop Preliminary Experiment II
Evolutionary Computation Meets Multiagent Systems for Better Solving Optimization Problems (CARVALHO; SICHMAN, 2019)	GEAR'18	ISBN: 978-981-13-6935-3 Book Chapter . Literature review
Applying Social Choice Theory to Solve Engineering Multi-Objective Optimization Problems (CARVALHO et al., 2020)	JCAES	ISSN: 2195-3880 Experiment II
Comparative analysis of selection hyper-heuristics for real-world multi-objective optimization problems (CARVALHO; ÖZCAN; SICHMAN, 2021)	Applied Sciences	ISSN: 2076-3417 Experiment III
Hyper-Heuristics based on Reinforcement Learning, Balanced Heuristic Selection and Group Decision Acceptance (JÚNIOR; ÖZCAN; CARVALHO, 2020)	Applied Soft Computing	ISSN: 1568-4946
Satisfying user preferences in optimised ridesharing services: A multi-agent multi-objective optimisation approach (CARVALHO; GOLPAYEGANI, 2022)	Applied Intelligence	ISSN: 1573-7497
Implementation	GitHub	github.com/vinixnan/MOABHH

REFERENCES

AARTS, E.; KORST, J. **Simulated Annealing and Boltzmann Machines: A Stochastic Approach to Combinatorial Optimization and Neural Computing**. New York, NY, USA: John Wiley & Sons, Inc., 1989. ISBN 0-471-92146-7.

ACAN, A.; LOTFI, N. A multiagent, dynamic rank-driven multi-deme architecture for real-valued multiobjective optimization. **Artificial Intelligence Review**, Springer Netherlands, Netherlands, p. 1–29, 2016.

ADRA, S. F. **Improving Convergence, Diversity and Pertinency in Multiobjective Optimisation**. Tese (Doutorado) — Department of Automatic Control and Systems Engineering, The University of Sheffield, 2007.

ALMEIDA, C. et al. Hyper-heuristics using multi-armed bandit models for multi-objective optimization. **Applied Soft Computing**, v. 95, p. 106520, 07 2020.

aO, W. A. et al. A multi-objective optimization approach for the integration and test order problem. **Information Science**, Elsevier Science Inc., New York, NY, USA, v. 267, p. 119–139, may 2014. ISSN 0020-0255.

AUER, P.; CESA-BIANCHI, N.; FISCHER, P. Finite-time analysis of the multiarmed bandit problem. **Machine Learning**, Kluwer Academic Publishers, Hingham, MA, USA, v. 47, n. 2-3, p. 235–256, maio 2002. ISSN 0885-6125.

AYDIN, M. E.; FOGARTY, T. C. Teams of autonomous agents for job-shop scheduling problems: An experimental study. **Journal of Intelligent Manufacturing**, v. 15, n. 4, p. 455–462, 2004.

BACK, T. Self-adaptation in genetic algorithms. In: **Proceedings of the First European Conference on Artificial Life**. Cambridge, MA.: MIT Press, 1992. p. 263–271.

BARBUCHA, D. A cooperative population learning algorithm for vehicle routing problem with time windows. **Neurocomputing**, v. 146, p. 210 – 229, 2014. ISSN 0925-2312. Bridging Machine learning and Evolutionary Computation (BMLEC) Computational Collective Intelligence.

BENNETT, K. P.; PARRADO-HERNÁNDEZ, E. The interplay of optimization and machine learning research. **J. Mach. Learn. Res.**, JMLR.org, v. 7, p. 1265–1281, dez. 2006. ISSN 1532-4435.

BORDA, J. C. de. Mémoire sur les élections au scrutin. **Histoire de l'Académie Royale des Sciences**, 1784.

BOUSSAID, I.; LEPAGNOT, J.; SIARRY, P. A survey on optimization metaheuristics. **Information Sciences**, v. 237, p. 82 – 117, 2013.

BRADSTREET, L. et al. Use of the WFG toolkit and PISA for comparison of MOEAs. In: **2007 IEEE Symposium on Computational Intelligence in Multi-Criteria Decision-Making**. Honolulu, USA: IEEE, 2007. p. 382–389.

BRAZDIL, P. et al. **Metalearning: Applications to Data Mining**. 1. ed. Berlin, Germany: Springer Publishing Company, Incorporated, 2008. ISBN 3540732624, 9783540732624.

BURKE, E. et al. A classification of hyper-heuristic approaches. In: **Handbook of metaheuristics**. Boston, MA: Springer, 2010. v. 146, p. 449–468.

BURKE, E. et al. Hyper-heuristics: An emerging direction in modern search technology. In: **Handbook of metaheuristics**. Boston, MA: Springer, 2003. p. 457–474.

BURKE, E. K. et al. Hyper-heuristics: a survey of the state of the art. **Journal of the Operational Research Society**, Palgrave Macmillan, v. 64, n. 12, p. 1695–1724, Dec 2013.

CADENAS, J. M.; GARRIDO, M. C.; MUNOZ, E. A cooperative system of metaheuristics. In: **7th International Conference on Hybrid Intelligent Systems (HIS 2007)**. Kaiserslautern, Germany: IEEE, 2007. p. 120–125.

CARVALHO, V. R. de. Uma hiper-heurística de seleção baseada em decomposição para estabelecer sequências de módulos para o teste de software. **Dissertação, Universidade Federal do Paraná (UFPR)**, 2015.

CARVALHO, V. R. de; GOLPAYEGANI, F. Satisfying user preferences in optimised ridesharing services:. **Applied Intelligence**, 2022.

CARVALHO, V. R. de et al. Applying social choice theory to solve engineering multi-objective optimization problems. **Journal of Control, Automation and Electrical Systems (JCAE)**, v. 31, n. 6, p. 119–128, fev. 2020. ISSN 2195-3899.

CARVALHO, V. R. de; SICHMAN, J. S. Applying Copeland Voting to Design an Agent-Based Hyper-heuristic. In: **Proceedings of the 16th Conference on Autonomous Agents and MultiAgent Systems**. São Paulo: IFAAMAS, 2017. p. 972–980.

CARVALHO, V. R. de; SICHMAN, J. S. Multi-Agent Election-Based Hyper-Heuristics. In: **Proc. of the 27th International Joint Conference on Artificial Intelligence**. Stockholm: IJCAI, 2018. p. 5779–5780.

CARVALHO, V. R. de; SICHMAN, J. S. Solving real-world multi-objective engineering optimization problems with an Election-Based Hyper-Heuristic. In: **OptMAS 2018: International Workshop on Optimisation in Multi-Agent Systems**. Stockholm: IJCAI, 2018.

CARVALHO, V. R. de; SICHMAN, J. S. Evolutionary computation meets multiagent systems for better solving optimization problems. In: KOCH, F. et al. (Ed.). **Evolutionary Computing and Artificial Intelligence: Essays Dedicated to Takao Terano on the Occasion of His Retirement**. Singapore: Springer Singapore, 2019. p. 27–41.

CARVALHO, V. R. de; VERGILIO, S. R.; POZO, A. Uma hiper-heurıstica de seleç ao de meta-heurısticas para estabelecer sequências de módulos para o teste de software. **Workshop de Engenharia de Software Baseada em Busca (WESB)**, 2015.

CARVALHO, V. R. de; ÖZCAN, E.; SICHMAN, J. S. Comparative analysis of selection hyper-heuristics for real-world multi-objective optimization problems. **Applied Sciences**, v. 11, n. 19, 2021. ISSN 2076-3417.

CASTRO, O. R.; POZO, A. Using hyper-heuristic to select leader and archiving methods for many-objective problems. In: GASPAR-CUNHA, A.; ANTUNES, C. H.; COELLO, C. C. (Ed.). **Evolutionary Multi-Criterion Optimization**. Cham: Springer International Publishing, 2015. p. 109–123. ISBN 978-3-319-15934-8.

CAUCHY, A. Méthode générale pour la résolution des systemes d'équations simultanées. **Comp. Rend. Sci. Paris**, v. 25, n. 1847, p. 536–538, 1847.

COELLO, C. **Evolutionary algorithms for solving multi-objective problems**. New York: Springer, 2007. ISBN 978-0-387-36797-2.

COPELAND, A. H. A reasonable social welfare function. In: **Mimeographed notes from a Seminar on Applications of Mathematics to the Social Sciences**. Michigan, USA: University of Michigan, 1951.

CORNELL, J. A. **Experiments with mixtures: designs, models, and the analysis of mixture data**. New Jersey, USA: John Wiley & Sons, 2011. v. 895.

COWLING, P.; KENDALL, G.; HAN, L. An investigation of a hyperheuristic genetic algorithm applied to a trainer scheduling problem. In: **Evolutionary Computation, 2002. CEC '02. Proceedings of the 2002 Congress on**. Honolulu, USA: IEEE Press, 2002. v. 2, p. 1185–1190.

COWLING, P. I.; KENDALL, G.; SOUBEIGA, E. A hyperheuristic approach to scheduling a sales summit. In: **Selected Papers from the Third International Conference on Practice and Theory of Automated Timetabling**. Berlin, Heidelberg: Springer Berlin Heidelberg, 2001. p. 176–190. ISBN 3-540-42421-0.

DEB, K.; JAIN, H. An evolutionary many-objective optimization algorithm using reference-point-based nondominated sorting approach, part i: Solving problems with box constraints. **IEEE Transactions on Evolutionary Computation**, IEEE Press., USA, v. 18, n. 4, p. 577–601, Aug 2014. ISSN 1089-778X.

DEB, K. et al. A fast and elitist multiobjective genetic algorithm: NSGA-II. **Evolutionary Computation, IEEE Transactions on**, IEEE Press., USA, v. 6, n. 2, p. 182–197, Apr 2002. ISSN 1089-778X.

DEB, K.; SUNDAR, J. Reference point based multi-objective optimization using evolutionary algorithms. In: **Proceedings of the 8th Annual Conference on Genetic and Evolutionary Computation**. New York, NY, USA: ACM, 2006. (GECCO '06), p. 635–642. ISBN 1-59593-186-4.

DEB, K. et al. Scalable test problems for evolutionary multiobjective optimization. In: **Evolutionary Multiobjective Optimization: Theoretical Advances and Applications**. London, UK: Springer London, 2005. p. 105–145. ISBN 978-1-84628-137-2.

DEMAZEAU, Y. From interactions to collective behaviour in agent-based systems. In: **In: Proceedings of the 1st. European Conference on Cognitive Science.** France: Saint-Malo, 1995. p. 117–132.

DEMAZEAU, Y.; MULLER, J.-P. (Ed.). **Decentralized A.I.: Proceedings of the European Workshop on Modelling Autonomous Agents in a Multi-Agent World, 1st, Cambridge, UK, 16- 18 Aug., 1989**. Amsterdam, The Netherlands, The Netherlands: Elsevier Science Publishers B. V., 1990. ISBN 0444887059.

DERRAC, J. et al. A practical tutorial on the use of nonparametric statistical tests as a methodology for comparing evolutionary and swarm intelligence algorithms. **Swarm and Evolutionary Computation**, v. 1, n. 1, p. 3 – 18, 2011. ISSN 2210-6502.

DIMITRIADOU, E.; WEINGESSEL, A.; HORNIK, K. A combination scheme for fuzzy clustering. **International Journal of Pattern Recognition and Artificial Intelligence**, World Scientific, v. 16, n. 07, p. 901–912, 2002.

DOMÍNGUEZ, X. S.; CARRIÉ, J. C. S.; PUJOL, F. A. Fuzzy clusterers combination by positional voting for robust document clustering. Sociedad Española para el Procesamiento del Lenguaje Natural, 2009.

DRAKE, J. H. et al. Recent advances in selection hyper-heuristics. **European Journal of Operational Research**, 2019. ISSN 0377-2217.

EIBEN, A. E. et al. Reinforcement learning for online control of evolutionary algorithms. In: **Proceedings of the 4th International Conference on Engineering Self-organising Systems**. Berlin, Germany: Springer, Berlin, Heidelberg, 2007. p. 151–160.

EIBEN, A. E.; MARCHIORI, E.; VALKÓ, V. A. Evolutionary algorithms with on-the-fly population size adjustment". In: **Parallel Problem Solving from Nature - PPSN VIII: 8th International Conference, Birmingham, UK, September 18-22, 2004. Proceedings**. Berlin, Germany: Springer Berlin Heidelberg, 2004. p. 41–50.

ELARBI, M. et al. Multi-objective optimization: Classical and evolutionary approaches. In: **Recent Advances in Evolutionary Multi-objective Optimization**. Berlin, Germany: Springer International Publishing, 2017. p. 1–30. ISBN 978-3-319-42978-6.

ENGAU, A. Domination and decomposition in multiobjective programming. 2007.

FEIGENBAUM, E. A.; FELDMAN, J. et al. **Computers and thought**. New York: AAAI Press, 1963.

FEO, T. A.; RESENDE, M. G. Greedy randomized adaptive search procedures. **Journal of global optimization**, Springer, Berlin, Germany, v. 6, n. 2, p. 109–133, 1995.

FONSECA, C. M.; FLEMING, P. J. Multiobjective optimization and multiple constraint handling with evolutionary algorithms. i. a unified formulation. **IEEE Transactions on Systems, Man, and Cybernetics: Part A**, IEEE Press, Piscataway, NJ, USA, v. 28, n. 1, p. 26–37, jan. 1998. ISSN 1083-4427.

FRITSCHE, G.; POZO, A. Cooperative based hyper-heuristic for many-objective optimization. In: **Proceedings of the Genetic and Evolutionary Computation Conference**. New York, NY, USA: ACM, 2019. (GECCO '19), p. 550–558. ISBN 978-1-4503-6111-8.

FRITSCHE, G.; POZO, A. The analysis of a cooperative hyper-heuristic on a constrained real-world many-objective continuous problem. In: **2020 IEEE Congress on Evolutionary Computation (CEC)**. Glasgow, UK: IEEE Press, 2020. p. 1–8.

GENDREAU, M.; POTVIN, J.-Y. **Handbook of Metaheuristics**. 2nd. ed. Berlin Heidelberg: Springer Publishing Company, Incorporated, 2010. ISBN 1441916636.

GHIASSI, M. et al. An application of multiple criteria decision making principles for planning machining operations. **IIE Transactions**, Taylor & Francis, v. 16, n. 2, p. 106–114, 1984.

GIOTIS, A. et al. Low-cost stochastic optimization for engineering applications. **Evolutionary Methods for Design Optimization and Control with Applications to Industrial Problems (EUROGEN-2001)**, p. 361–366, 2001.

GOH, C. K.; TAN, K. C. A competitive-cooperative coevolutionary paradigm for dynamic multiobjective optimization. **IEEE Transactions on Evolutionary Computation**, IEEE Press., USA, v. 13, n. 1, p. 103–127, Feb 2009. ISSN 1089-778X.

GOLDBERG, D. E.; HOLLAND, J. H. Genetic algorithms and machine learning. **Machine learning**, Springer Netherlands, v. 3, n. 2, p. 95–99, 1988.

GOLINSKI, J. Optimal synthesis problems solved by means of nonlinear programming and random methods. **Journal of Mechanisms**, v. 5, n. 3, p. 287 – 309, 1970. ISSN 0022-2569.

GONÇALVES, R. A.; ALMEIDA, C. P.; POZO, A. Upper confidence bound (ucb) algorithms for adaptive operator selection in moea/d. In: **Evolutionary Multi-Criterion Optimization**. Berlin, Germany: Springer International Publishing, 2015, (Lecture Notes in Computer Science). p. 411–425.

GONÇALVES, R. A. et al. Decomposition based multiobjective hyper heuristic with differential evolution. In: **Computational Collective Intelligence**. Berlin, Germany: Springer International Publishing, 2015, (Lecture Notes in Computer Science). p. 129–138.

GUIZZO, G. et al. A hyper-heuristic for the multi-objective integration and test order problem. In: **Proceedings of the 2015 Annual Conference on Genetic and Evolutionary Computation**. New York, NY, USA: ACM, 2015. (GECCO '15), p. 1343–1350. ISBN 978-1-4503-3472-3.

GUIZZO, G.; VERGILIO, S. R.; POZO, A. T. R. Evaluating a multi-objective hyper-heuristic for the integration and test order problem. In: **2015 Brazilian Conference on Intelligent Systems (BRACIS)**. Natal, Brazil: SBC, 2015. p. 1–6.

GUNAWAN, S.; AZARM, S. Multi-objective robust optimization using a sensitivity region concept. **Structural and Multidisciplinary Optimization**, v. 29, n. 1, p. 50–60, Jan 2005. ISSN 1615-1488.

GUNAWAN, S.; FARHANG-MEHR, A.; AZARM, S. Multi-level multi-objective genetic algorithm using entropy to preserve diversity. In: **International Conference on Evolutionary Multi-Criterion Optimization**. Berlin Heidelberg: Springer, 2003. p. 148–161.

GUTIN, G.; PUNNEN, A. P. **The traveling salesman problem and its variations**. Berlin, Germany: Springer Science & Business Media, 2006. v. 12.

HANSEN, M. P. Use of substitute scalarizing functions to guide a local search based heuristic: The case of motsp. **Journal of Heuristics**, Springer, Berlin Heidelberg, v. 6, n. 3, p. 419–431, 2000.

HANSEN, M. P. et al. **Evaluating the Quality of Approximations to the Non-Dominated Set**. 1998.

HARIK, G. R.; LOBO, F. G. A parameter-less genetic algorithm. In: **Proceedings of the 1st Annual Conference on Genetic and Evolutionary Computation - Volume 1**. [S.l.]: Morgan Kaufmann Publishers Inc., 1999. (GECCO'99), p. 258–265. ISBN 1-55860-611-4.

HARMAN, M.; JONES, B. F. Search-based software engineering. **Information and Software Technology**, v. 43, n. 14, p. 833 – 839, 2001. ISSN 0950-5849.

HINTERDING, R.; MICHALEWICZ, Z.; PEACHEY, T. C. Self-adaptive genetic algorithm for numeric functions. In: **Parallel Problem Solving from Nature — PPSN IV: International Conference on Evolutionary Computation — The 4th International Conference on Parallel Problem Solving from Nature Berlin, Germany, September 22–26, 1996 Proceedings**. Berlin, Germany: Springer Berlin Heidelberg, 1996. p. 420–429.

HUBAND, S. et al. A review of multiobjective test problems and a scalable test problem toolkit. **Evolutionary Computation, IEEE Transactions on**, IEEE Press., USA, p. 477–506, 2006.

IFN. **Black Box Optimization Competition, EMO'2017 Real-World Problems**. 2017. Disponível em: <https://www.ini.rub.de/PEOPLE/glasmtbl/projects/bbcomp/>.

ISHIBUCHI, H. et al. Comparison of hypervolume, igd and igd+ from the viewpoint of optimal distributions of solutions. In: DEB, K. et al. (Ed.). **Evolutionary Multi-Criterion Optimization**. Cham: Springer International Publishing, 2019. p. 332–345.

ISHIBUCHI, H. et al. Modified distance calculation in generational distance and inverted generational distance. In: GASPAR-CUNHA, A.; ANTUNES, C. H.; COELLO, C. C. (Ed.). **Evolutionary Multi-Criterion Optimization**. Berlin Heidelberg: Springer International Publishing, 2015. p. 110–125. ISBN 978-3-319-15892-1.

JAIN, H.; DEB, K. An evolutionary many-objective optimization algorithm using reference-point based nondominated sorting approach, part ii: Handling constraints and extending to an adaptive approach. **IEEE Transactions on Evolutionary Computation**, IEEE Press., USA, v. 18, n. 4, p. 602–622, 2014.

JIANG, S. et al. Consistencies and contradictions of performance metrics in multiobjective optimization. **IEEE Transactions on Cybernetics**, IEEE Press., USA, v. 44, n. 12, p. 2391–2404, Dec 2014.

JÚNIOR, V. A. d. S.; ÖZCAN, E.; CARVALHO, V. R. de. Hyper-heuristics based on reinforcement learning, balanced heuristic selection and group decision acceptance. **Applied Soft Computing**, v. 97, p. 106760, 2020. ISSN 1568-4946.

KARAFOTIAS, G.; HOOGENDOORN, M.; EIBEN, A. Evaluating reward definitions for parameter control. In: **Applications of Evolutionary Computation**. Berlin, Germany: Springer International Publishing, 2015, (Lecture Notes in Computer Science). p. 667–680.

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

KELLY, J. S. **Social choice theory: An introduction**. Berlin, Germany: Springer Science & Business Media, 2013.

KEMENY, J. G. Mathematics without numbers. **Daedalus**, The MIT Press, v. 88, n. 4, p. 577–591, 1959. ISSN 00115266.

KHEIRI, A.; KEEDWELL, E. A hidden markov model approach to the problem of heuristic selection in hyper-heuristics with a case study in high school timetabling problems. **Evolutionary computation**, MIT Press, v. 25, n. 3, p. 473–501, 2017.

KNOWLES, J.; CORNE, D. On metrics for comparing nondominated sets. In: **Evolutionary Computation, 2002. CEC '02. Proceedings of the 2002 Congress on**. Honolulu, USA: IEEE Press., 2002. v. 1, p. 711–716.

KOZA, J. R. Evolution of subsumption using genetic programming. In: **Proceedings of the First European Conference on Artificial Life**. Amsterdam': Elsevier Science Bv, 1992. p. 110–119.

KUKKONEN, S.; LAMPINEN, J. Gde3: The third evolution step of generalized differential evolution. In: **Evolutionary Computation, 2005. The 2005 IEEE Congress on**. USA: IEEE Press., 2005. v. 1, p. 443–450.

LI, B. et al. Many-objective evolutionary algorithms: A survey. **ACM Comput. Surv.**, ACM, New York, NY, USA, v. 48, n. 1, p. 13:1–13:35, set. 2015. ISSN 0360-0300.

LI, K. et al. An Evolutionary Many-Objective Optimization Algorithm Based on Dominance and Decomposition. **IEEE Transactions on Evolutionary Computation**, IEEE Press., USA, v. 19, n. 5, p. 694–716, oct 2015. ISSN 1089-778X.

LI, K. et al. Adaptive operator selection with bandits for a multiobjective evolutionary algorithm based on decomposition. **IEEE Transactions on Evolutionary Computation**, IEEE Press., USA, v. 18, n. 1, p. 114–130, Feb 2014. ISSN 1089-778X.

LI, W.; ÖZCAN, E.; JOHN, R. Multi-objective evolutionary algorithms and hyper-heuristics for wind farm layout optimisation. **Renewable Energy**, v. 105, p. 473 – 482, 2017. ISSN 0960-1481.

Li, W.; ÖZCAN, E.; John, R. A learning automata-based multiobjective hyper-heuristic. **IEEE Transactions on Evolutionary Computation**, IEEE Press., USA, v. 23, n. 1, p. 59–73, Feb 2019. ISSN 1089-778X.

Li, W. et al. A modified indicator-based evolutionary algorithm (mIBEA). In: **2017 IEEE Congress on Evolutionary Computation (CEC)**. San Sebastián, Spain: IEEE Press., 2017. p. 1047–1054.

LIAO, X. et al. Multiobjective optimization for crash safety design of vehicles using stepwise regression model. **Structural and multidisciplinary optimization**, Springer Berlin Heidelberg, v. 35, n. 6, p. 561–569, 2008.

LITTLE, I. M. Social choice and individual values. **Journal of Political Economy**, The University of Chicago Press, v. 60, n. 5, p. 422–432, 1952.

MAASHI, M.; ÖZCAN, E.; KENDALL, G. A multi-objective hyper-heuristic based on choice function. **Expert Systems with Applications**, Elsevier, v. 41, n. 9, p. 4475–4493, 2014.

MALEK, R. An agent-based hyper-heuristic approach to combinatorial optimization problems. In: **2010 IEEE International Conference on Intelligent Computing and Intelligent Systems**. USA: IEEE Press., 2010. v. 3, p. 428–434.

MAO, A.; PROCACCIA, A. D.; CHEN, Y. Better human computation through principled voting. In: **Proc. of the Twenty-Seventh AAAI Conference on Artificial Intelligence, July 14-18, 2013**. Bellevue, Washington, USA: AAAI, 2013.

MARTIN, S. et al. A multi-agent based cooperative approach to scheduling and routing. **European Journal of Operational Research**, v. 254, n. 1, p. 169 – 178, 2016. ISSN 0377-2217.

MCCLYMONT, K. et al. A general multi-objective hyper-heuristic for water distribution network design with discolouration risk. **Journal of Hydroinformatics**, IWA Publishing, v. 15, n. 3, p. 700–716, 2013.

MEIGNAN, D.; KOUKAM, A.; CRÉPUT, J.-C. Coalition-based metaheuristic: a self-adaptive metaheuristic using reinforcement learning and mimetism. **Journal of Heuristics**, v. 16, n. 6, p. 859–879, Dec 2010. ISSN 1572-9397.

MILANO, M.; ROLI, A. Magma: a multiagent architecture for metaheuristics. **IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)**, IEEE Press., USA, v. 34, n. 2, p. 925–941, 2004.

MOČKUS, J. On bayesian methods for seeking the extremum. In: MARCHUK, G. I. (Ed.). **Optimization Techniques IFIP Technical Conference: Novosibirsk, July 1–7, 1974**. Berlin, Heidelberg: Springer Berlin Heidelberg, 1975. p. 400–404. ISBN 978-3-662-38527-2.

MOULIN, H. Condorcet's principle implies the no show paradox. **Journal of Economic Theory**, v. 45, n. 1, p. 53–64, 1988. ISSN 0022-0531.

NEBRO, A. J. et al. SMPSO: A new PSO-based metaheuristic for multi-objective optimization. In: **2009 IEEE Symposium on Computational Intelligence in Multi-Criteria Decision-Making(MCDM)**. USA: IEEE Press., 2009. p. 66–73.

NEBRO, A. J.; DURILLO, J. J.; VERGNE, M. Redesigning the jmetal multi-objective optimization framework. In: **Proceedings of the Companion Publication of the 2015 Annual Conference on Genetic and Evolutionary Computation**. New York, NY, USA: ACM, 2015. (GECCO Companion '15), p. 1093–1100. ISBN 978-1-4503-3488-4.

NOTTINGHAM, U. of. **The hyper-heuristic framework featuring the domain barrier**. 2011. Disponível em: <http://www.asap.cs.nott.ac.uk/external/chesc2011/hyflex_description.html>.

NUGRAHENI, C. E.; ABEDNEGO, L. Multi agent hyper-heuristics based framework for production scheduling problem. In: **2016 International Conference on Informatics and Computing (ICIC)**. USA: IEEE Press., 2016. p. 309–313.

NURMI, H. **Voting Systems for Social Choice**. Dordrecht, Netherlands: Springer Netherlands, 2010. 167–182 p. ISBN 978-90-481-9097-3.

OSMAN, I. H.; LAPORTE, G. **Metaheuristics: A bibliography**. Berlin, Germany: Springer, 1996.

OUELHADJ, D.; PETROVIC, S. A cooperative hyper-heuristic search framework. **Journal of Heuristics**, v. 16, n. 6, p. 835–857, Dec 2010. ISSN 1572-9397.

PAPPA, G. L. et al. Contrasting meta-learning and hyper-heuristic research: the role of evolutionary algorithms. **Genetic Programming and Evolvable Machines**, v. 15, n. 1, p. 3–35, Mar 2014. ISSN 1573-7632.

PAQUETE, L.; CHIARANDINI, M.; STÜTZLE, T. Pareto local optimum sets in the biobjective traveling salesman problem: An experimental study. In: GANDIBLEUX, X. et al. (Ed.). **Metaheuristics for Multiobjective Optimisation**. Berlin, Heidelberg: Springer Berlin Heidelberg, 2004. p. 177–199. ISBN 978-3-642-17144-4.

PEARL, J. Heuristics: intelligent search strategies for computer problem solving. Addison-Wesley Pub. Co., Inc., Reading, MA, 1984.

POLONI, C.; MOSETTI, G.; CONTESSI, S. Multi objective optimization by gas: Application to system and component design. In: JOHN WILEY & SONS, LTD. **ECCOMAS'96: Computational Methods in Applied Sciences' 96**. New Jersey, USA, 1996. p. 1–7.

POMEROL, J.; BARBA-ROMERO, S. **Multicriterion decision in management: principles and practice**. Berlin, Germany: Springer Science & Business Media, 2012. v. 25.

PSYCHAS, I.-D.; DELIMPASI, E.; MARINAKIS, Y. Hybrid evolutionary algorithms for the multiobjective traveling salesman problem. **Expert Syst. Appl.**, Pergamon Press, Inc., Tarrytown, NY, USA, v. 42, n. 22, p. 8956–8970, dez. 2015. ISSN 0957-4174.

QUAGLIARELLA, D.; VICINI, A. Sub-population policies for a parallel multiobjective genetic algorithm with applications to wing design. In: **SMC'98 Conference Proceedings. 1998 IEEE International Conference on Systems, Man, and Cybernetics (Cat. No.98CH36218)**. USA: IEEE Press., 1998. v. 4, p. 3142–3147 vol.4. ISSN 1062-922X.

RABAK, C. S.; SICHMAN, J. S. Using a-teams to optimize automatic insertion of electronic components. **Adv. Eng. Informatics**, v. 17, n. 2, p. 95–106, 2003.

RICCI, A.; VIROLI, M.; OMICINI, A. Programming mas with artifacts. In: **Proceedings of the Third International Conference on Programming Multi-Agent Systems**. Berlin, Heidelberg: Springer-Verlag, 2006. (ProMAS'05), p. 206–221. ISBN 3-540-32616-2, 978-3-540-32616-8.

RICCI, A. and VIROLI, M. and OMICINI, A. Give agents their artifacts: the a&a approach for engineering working environments in MAS. In: DURFEE, E. H. et al. (Ed.). **6th International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS 2007)**. Honolulu, Hawaii, USA: IFAAMAS, 2007. p. 150. ISBN 978-81-904262-7-5.

RICE, J. R. The algorithm selection problem. In: RUBINOFF, M.; YOVITS, M. C. (Ed.). USA: Elsevier, 1976, (Advances in Computers, v. 15). p. 65 – 118.

RUDER, S. An overview of gradient descent optimization algorithms. **arXiv preprint arXiv:1609.04747**, 2016.

RUSSELL, S.; NORVIG, P. A modern approach. Prentice Hall, v. 3, 2010.

SABAR, N. et al. The automatic design of hyper-heuristic framework with gene expression programming for combinatorial optimization problems. **IEEE TRANSACTIONS ON EVOLUTIONARY COMPUTATION**, IEEE Press., USA, 2014.

SABAR, N. et al. A dynamic multiarmed bandit-gene expression programming hyper-heuristic for combinatorial optimization problems. **IEEE Transactions on Cybernetics**, IEEE Press., USA, v. 45, n. 2, p. 217–228, Fevereiro 2015.

SEVILLANO, X.; ALÍAS, F.; SOCORÓ, J. C. Bordaconsensus: a new consensus function for soft cluster ensembles. In: **In Proc. 30th ACM SIGIR Conference**. USA: ACM Press., 2007. p. 743–744.

SMITH-MILES, K. Measuring instance difficulty for combinatorial optimization problems. **Computers & Operations Research**, v. 39, n. 5, p. 875 – 889, 2012. ISSN 0305-0548.

SMITH-MILES, K. et al. Understanding the relationship between scheduling problem structure and heuristic performance using knowledge discovery, lncs. 01 2009.

SMORODKINA, E.; TAURITZ, D. Greedy population sizing for evolutionary algorithms. In: **2007 IEEE Congress on Evolutionary Computation**. USA: IEEE Press., 2007. p. 2181–2187.

SRINIVAS, M.; PATNAIK, L. M. Adaptive probabilities of crossover and mutation in genetic algorithms. **IEEE Transactions on Systems, Man, and Cybernetics**, IEEE Press., USA, v. 24, n. 4, p. 656–667, Apr 1994. ISSN 0018-9472.

SRINIVAS, N.; DEB, K. Multiobjective optimization using nondominated sorting in genetic algorithms. **Evolutionary Computation, IEEE Transactions on**, IEEE Press., USA, v. 2, p. 221–248, 1994.

STADLER, W.; DAUER, J. Multicriteria optimization in engineering: A tutorial and survey. **Structural optimization: Status and promise**, AIAA: Washington, DC, v. 150, 1993.

STORN, R.; PRICE, K. Differential evolution – a simple and efficient heuristic for global optimization over continuous spaces. **Journal of Global Optimization**, v. 11, n. 4, p. 341–359, Dec 1997. ISSN 1573-2916.

SUN, S. et al. **A Survey of Optimization Methods from a Machine Learning Perspective**. 2019.

TALBI, E.; BACHELET, V. COSEARCH: A Parallel Cooperative Metaheuristic. **Journal of Mathematical Modelling and Algorithms**, v. 5, n. 1, p. 5–22, 2006.

TALUKDAR, S. et al. Asynchronous teams: Cooperation schemes for autonomous agents. **Journal of Heuristics**, Kluwer Academic Publishers, Hingham, MA, USA, v. 4, n. 4, p. 295–321, dez. 1998. ISSN 1381-1231.

TAN, K. C.; LEE, T.; KHOR, E. Evolutionary algorithms for multi-objective optimization: Performance assessments and comparisons. **Artificial Intelligence Review**, v. 17, n. 4, p. 251–290, 2002.

TAPABRATA, R.; KANG, T.; SEOW, K. C. Multiobjective design optimization by an evolutionary algorithm. **Engineering Optimization**, Taylor & Francis, v. 33, n. 4, p. 399–424, 2001.

TRAN, R. et al. Fast and effective multi-objective optimisation of wind turbine placement. In: **Proceedings of the 15th Annual Conference on Genetic and Evolutionary Computation**. New York, NY, USA: ACM, 2013. (GECCO '13), p. 1381–1388. ISBN 978-1-4503-1963-8.

VAFAEE, F.; NELSON, P. C. An explorative and exploitative mutation scheme. In: **IEEE Congress on Evolutionary Computation**. USA: IEEE Press., 2010. p. 1–8. ISSN 1089-778X.

VÁZQUEZ-RODRÍGUEZ, J.; PETROVIC, S. A mixture experiments multi-objective hyper-heuristic. **Journal of the Operational Research Society**, Palgrave Macmillan, v. 64, n. 11, p. 1664–1675, 2012.

VELDHUIZEN, D. A. V. **Multiobjective Evolutionary Algorithms: Classifications, Analyses, and New Innovations**. Tese (Doutorado) — Air Force Institute of Technology, Wright Patterson AFB, OH, USA, 1999. AAI9928483.

VRUGT, J. A.; ROBINSON, B. A. Improved evolutionary optimization from genetically adaptive multimethod search. **Proceedings of the National Academy of Sciences**, National Academy of Sciences, v. 104, n. 3, p. 708–711, 2007. ISSN 0027-8424.

WHILE, L.; BRADSTREET, L.; BARONE, L. A fast way of calculating exact hypervolumes. **IEEE Transactions on Evolutionary Computation**, IEEE Press., USA, v. 16, n. 1, p. 86–95, Feb 2012. ISSN 1089-778X.

WOLPERT, D. H.; MACREADY, W. G. No free lunch theorems for optimization. **IEEE Transactions on Evolutionary Computation**, IEEE Press., USA, v. 1, n. 1, p. 67–82, Apr 1997. ISSN 1089-778X.

WOOLDRIDGE, M. **An introduction to multiagent systems**. New Jersey, USA: John Wiley & Sons, 2009.

YANG, X.-S. Multiobjective firefly algorithm for continuous optimization. **Engineering with Computers**, v. 29, n. 2, p. 175–184, Apr 2013. ISSN 1435-5663.

YEN, G. G.; HE, Z. Performance metric ensemble for multiobjective evolutionary algorithms. **IEEE Transactions on Evolutionary Computation**, IEEE Press., USA, v. 18, n. 1, p. 131–144, Feb 2014. ISSN 1089-778X.

ZHANG, Q.; LI, H. MOEA/D: A multiobjective evolutionary algorithm based on decomposition. **IEEE Transactions on Evolutionary Computation**, IEEE Press., USA, Dec 2007.

ZHANG, Q. et al. Multiobjective optimization test instances for the cec 2009 special session and competition. **University of Essex, Colchester, UK and Nanyang technological University, Singapore, special session on performance assessment of multi-objective optimization algorithms, technical report**, p. 1–30, 2008.

ZITZLER, E.; DEB, K.; THIELE, L. Comparison of multiobjective evolutionary algorithms: Empirical results. **Evol. Comput.**, MIT Press, p. 173–195, Junho 2000.

ZITZLER, E.; KÜNZLI, S. Indicator-based selection in multiobjective search. In: **PPSN**. Berlin, Germany: Springer, 2004. (Lecture Notes in Computer Science, v. 3242), p. 832–842. ISBN 3-540-23092-0.

ZITZLER, E.; LAUMANNS, M.; THIELE, L. SPEA2: Improving the strength pareto evolutionary algorithm for multiobjective optimization. In: **Evolutionary Methods for Design Optimization and Control with Applications to Industrial Problems**. Barcelona, Spain: International Center for Numerical Methods in Engineering, 2001. p. 95–100.

ZITZLER, E.; THIELE, L. Multiobjective evolutionary algorithms: A comparative case study and the strength pareto approach. **IEEE Transactions on Evolutionary Computation**, IEEE Press., USA, v. 3, n. 4, p. 257–271, nov 1999. ISSN 1089-778X.

ZITZLER, E. et al. Performance assessment of multiobjective optimizers: An analysis and review. **IEEE Transactions on Evolutionary Computation**, IEEE Press, Piscataway, NJ, USA, v. 7, n. 2, p. 117–132, abr. 2003. ISSN 1089-778X.