Instituto Tecnológico y de Estudios Superiores de Monterrey

Monterrey Campus

School of Engineering and Sciences

Feature Transformations for Improving the Performance of Selection Hyper-heuristics on Job Shop Scheduling Problems

A dissertation presented by

Fernando Garza Santisteban

Submitted to the
School of Engineering and Sciences
in partial fulfillment of the requirements for the degree of

Master of Science

in

Computer Science

Monterrey, Nuevo León, May, 2019

Declaration of Authorship

I, Fernando Garza Santisteban, declare that this dissertation entitled *Feature Transformations for Improving the Performance of Selection Hyper-heuristics on Job Shop Scheduling Problems*, and the work presented in it are my own. I also confirm that:

- This work was done wholly or mainly while in candidature for a research degree at this University.
- Where any part of this dissertation has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated.
- Where I have consulted the published work of others, this is always clearly attributed.
- Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this dissertation is entirely my own work.
- I have acknowledged all main sources of help.
- Where the dissertation is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself.

Fernando Garza Santisteban
Monterrey, Nuevo León, May, 2019

Dedication

To God, my family, and friends.

Thanks for all your support, patience and reassurance throughout the development of this research.

Acknowledgements

This research would not have been possible without the help of my advisors. I would like to praise the instructional and supportive efforts of professors Terashima, Amaya and Ortiz.
I am very thankful for your time and dedication.

A special mention is due to professor Taillard, which has kindly provided the source code of an instance generator used for this research.

Also, thankfulness is due to both Tecnológico de Monterrey for their support on tuition and CONACyT for the economic scholarship provided throughout my studies.

Feature Transformations for Improving the Performance of Selection Hyper-heuristics on Job Shop Scheduling Problems

by

Fernando Garza Santisteban

Abstract

Solving Job Shop (JS) scheduling problems is a hard combinatorial optimization problem. Nevertheless, it is one of the most present problems in real-world scheduling environments. Throughout the recent computer science history, a plethora of methods to solve this problem have been proposed. Despite this fact, the JS problem remains a challenge. The domain itself is of interest for the industry and also many operations research problems are based on this problem. The solution to JS problems is overall beneficial to the industry by generating more efficient processes. Authors have proposed solutions to this problem using dispatching rules, direct mathematical methods, meta-heuristics, among others. In this research, the application of feature transformations for the generation of improved selection constructive hyper-heuristics (HHs) is shown. There is evidence that applying feature transformations on other domains has produced promising results; Also, no previous work was found where this approach has been used for the JS domain.

This thesis is presented to earn the Master's degree in Computer Science of Tecnológico de Monterrey. The research's main goals are: (1) the assessment of the extent to which HHs can perform better on JS problems than single heuristics, and that they are not specific to the instances used to train them; and (2), the degree to which HHs generated with feature transformations are revamped. Experiments were carried out using instances of various sizes published in the literature. The research involved profiling the set of heuristics chosen, analyzing the interactions between the heuristics and feature values throughout the construction of a solution, and studying the performance of HHs without transformations and by using two transformations found in the literature. Results indicate that for the instances used, HHs were able to outperform the results achieved by single heuristics. Regarding feature transformations, it was found that they induce a scaling effect to feature values throughout the solution process, which produces more stable HHs, with a median performance comparable to HHs without feature transformations, but not necessarily better. Results are conclusive in terms of the objectives of this research. Nevertheless, there are several ideas that could be explored to improve the HHs, which are outlined and discussed in the final Chapter of the thesis.

The following major contributions are derived from this research: (1) applying a selection constructive HH approach, with feature transformations, to the JS domain; (2) the rationale behind the JS subproblem dependance in terms of the solution paths followed by the heuristics, which has a great impact in the training process of the HHs; (3) a method to determine the most suitable parameters to apply feature transformations, which could be extended for other domains of combinatorial optimization problems; and (4) a framework for studying HHs in the Job Shop domain.

List of Figures

List of Tables

Contents

Chapter 1

Introduction

In the following sections, a general description of the importance of the problem to be solved is presented. Also, the Job Shop scheduling problem is introduced with a brief explanation of several approaches to solving this kind of optimization problems. Next, a description of how hyper-heuristics and feature transformations will be used to tackle the problem is presented. After this, the hypothesis and objectives of the thesis research are described, ending with the main contributions derived from this work and how this document is organized.

1.1 Motivations

Scheduling systems have been very popular in supply chain based industries, where the process of transforming materials into products involves several processing units with distinct processing times. It is fair to say that one of their main objectives is to make the overall production process more efficient, which in the end translates to less overhead and production costs. Although scheduling problems are sometimes related to applications to manufacturing industries, in general, any process which involves activities and times to carry them out can be grouped in the scheduling domain. Hence, it is relevant to find methods which help to find optimal solutions to scheduling problems.

The automatic process of optimizing a schedule has been a topic of study since the 1950s [74]. Throughout the years, the problems have been separated and classified into different categories, depending on the qualities and quantities of the problem to be solved. One of such categories corresponds to problems where the aim is to schedule a set of jobs on a set of machines, subject to the constraint that each machine can handle at most one job at a time, and the characteristic that each job has a specified processing order through the machines. The objective is to schedule the jobs in such a way that the maximum of their completion times, which is known as the make-span [5] is minimized. This problem is generally referred to as the Job Shop Scheduling (JSS) problem.

JSS problems are quite difficult to handle in real-world applications, in fact, the domain belongs to the non-deterministic polynomial time (NP-hard) problem class. This relation places JSS into the class of problems that are not solvable in polynomial time (called P problems). In practice, there are $j! * m$ possibilities to schedule for the j number of jobs and m number of machines on a given shop floor [88].

When an exhaustive search for an optimal solution is unfeasible, a common approach is to use rule-of-thumb decisions, commonly known as heuristics. This technique is used for approximating the solution to a problem instead of finding the exact solution. Heuristics are widely used in NP problems because they produce a solution in a reasonable time frame, and their efficiency could be improved when used in conjunction with other optimization algorithms [96].

Nevertheless, heuristics are very sensitive to problem features, so small changes in the problem state can significantly affect their performance. To overcome this, we propose to implement a hyper-heuristic (HH) based method. This solution model involves selecting several heuristics during the search. In general, it has been shown for several domains [7] that HHs can use the most suitable heuristic for each step of the search and also have the ability to be applicable to a range of different instances and problems. Depending on the process of using the low-level heuristics and the function of the HH in a problem, HHs can be classified into four classes [93]. In terms of the process of using low-level heuristics, HHs are classified as selection HHs if they select heuristics from a set of previously defined ones. On the contrary, they are classified as generation HHs if they produce new rules. In terms of the role of the HH in regards to the problem, a HH is considered perturbative if its main goal is to modify a given solution to improve it and constructive if its goal is to generate a new solution from scratch. Hence, HHs are classified as generation constructive, generation perturbative, selection constructive, or selection perturbative. An extensive review of this classification is provided in Chapter 2. This research is focused on using selection constructive HHs to solve the Job Shop scheduling problem.

Furthermore, this research lies within the context of the previous work done by Amaya et al. [4]. The authors have shown that, when training similar HHs to the ones being used in this research but in other optimization domains such as CSP and Knapsack, the application of transformations on the characteristics of a current state of the problem's solution have produced better results than the ones produced when no transformations were applied. In this research, we propose to train constructive HHs with a Simulated Annealing (SA) approach, and also apply feature transformations to the problem state in order to study the effect of the transformations on the JS domain.

1.2 The JSS problem

A set of n jobs $j_1, j_2, ..., j_n$ on a set of m machines $m_1, m_2, ..., m_m$ are given, with the order sequence for each job on the set of machines. Each job has to be processed sequentially on one or more machines, having different processing times for each case. Such cases can be defined as tasks or activities a_{ik} for each job j_i at machine m_k. Thus, each job j_i consists of a set of at most m activities $a_{i1}, a_{i2}, ..., a_{im}$ where the activity's time $p_{a_{ik}}$ is different for each machine k. The JSS problem consists in scheduling each job j_i at each machine m_k, that is, determining the starting time of every activity of each job, subject to some constraints.

There are three constraints involved in the solution of the JSS problem [84]:

1. Precedence: no activity can be scheduled to start or end before another activity of the same job which has higher precedence.

2. Capacity: every machine can process one activity at a given time.

3. Non-preemption: if any activity starts at a machine, it has to finish before another activity can start on that same machine.

The objective is to generate a schedule so that the time at which the last activity of the whole schedule finishes (*makespan*) is minimum.

1.2.1 Survey of Related Solution Approaches

Many solution methods to JSS problems have been proposed throughout the history of operations research. Kurdi [69] provides an overview of some methods used for solving the JSS that include an enumerative approach implemented by Lageweg, Lenstra and Rinooy [70], and the branch and bound method used by Carlier and Pinson [25]. These methods worked very well for small instances, however, when increasing the instance size their computational cost increased very fast. Such problems caused research efforts to shift towards heuristic approaches. Examples of these were shown earlier by the dispatching rules proposed by Blackstone, Philips and Hogg [55]. Also, with the Shifting Bottleneck procedure, developed by Adams, Balas and Zawack [2]. These methods although computationally efficient, provide solutions for which quality is not guaranteed.

Although the main classes of solution methods are described in the next chapter, in Section 2.3, trends of recent research has been focused in using metaheuristics to try to find solutions to the JSSP without exerting traditional optimization methods. Below are some examples that illustrate these recent metaheuristics.

A recent state-of-the-art work that involves an evolutionary approach is the work from Zhao, Zhang, and Wang [121], which propose a modification of Shuffled Complex Evolution (SCE), called Improved SCE (ISCE). SCE is an evolutionary algorithm that evolves locally as traditional algorithms (i.e. genetic algorithms) do but also evolves globally by altering candidates in parallel local searches. The authors improve such strategy by including a methodology to overcome stagnation. It is shown that the ISCE algorithm is more effective than the basic SCE algorithm in the quality of solution and rate of convergence. However, the performance of the ISCE algorithm on the large-scale Job Shop scheduling problem fluctuated in different situations, especially when the scale of the JJS problem increases. Other similar works include the one by Wang, Cai and Li [112], that provide an adaptive multi-population genetic algorithm; and Kurdi [69], which uses an Island Model Genetic Algorithm (IMGE). In general, genetic-based algorithms are abundant in the literature.

Metaheuristics have also been used for solving the JSS problem. For example, SA algorithms are used in the works by Satake, Morikawa, Takahashi and Nakamura [101], and by Laarhoven, Aarts and Lenstra [110]. Traditional Tabu Search is present in the research made by Nowicki and Smutnicki [89] and Zhang, Li, Guan and Rao [120]. Also, an improvement of the aformentioned method is the one proposed by Peng, Lü and Cheng [92], which use a Tabu Search/Path Relinking (TS/PR) technique that is able to improve the upper bounds for 49 of the 205 instances in which it was tested.

Other traditional meta-heuristics include Particle Swarm Optimization (PSO) by Lian, Jiao and Gu [75], and by Lin et al. [77]; and more recently by Gao et al. [46], in which the authors propose a hybrid Particle Swarm Optimization (PSO) algorithm based on the Variable

Neighborhood Search (VNS) algorithm. Also, Ant Colony Optimization (ACO), exemplified by Blum and Samples [15], and Seo and Kim [103] Cheng, Peng and Lü.

Most recent research tends to be a mixture between other metaheuristic approaches. An example of this is proposed by Cheng et al. [30], where they integrate TS into the framework of an EA for what they called a Hybrid Evolutionary Algorithm.

During the development of this research, a conference article by Chaurasia et al. [27] was published. This was the only evidence found of similar research to the one produced by this thesis research. In the said study, the authors use a genetic-based evolutionary algorithm to generate and fit HHs to constructively solve the JSSP. Nevertheless, in the work mentioned, the low-level heuristics work directly in the offspring generated by the algorithm and differ from the ones in this research in the way that the heuristics used herein are rules that act over the set of jobs. Despite this, it is a work worth mentioning because uses a similar type of HHs to the ones proposed here for generating solutions to the JSSP.

Finally, no work was found that uses either a HH model such as the one proposed in this work, nor the techniques described by Amaya et al. [4] for applying feature transformations to improve HHs on the Job Shop domain.

1.2.2 Hyper-heuristic Formulation

As stated in the previous section, several low-level heuristics can be used in constructing or modifying a schedule. The HH model used throughout this research is focused on applying different constructive heuristics for each state of the JSSP. A state S_t of the problem is defined as the schedule of the problem at the specific moment t. Hence, the job of the HH is to provide a specific heuristic h to use at each S_t of the schedule. Every S_t is represented as the set of values of the predefined features in the problem for each state. The HH is comprised of blocks that represent pairs of states and actions to be applied. The S_t is compared to each block of the HH and the action is chosen accordingly. A detailed description of this formulation is furtherly described in Section 3.1.

1.2.3 Feature Transformations

There exists evidence for different optimization problem domains such as CSP and Knapsack [3, 4] that using features without any transformation during the training phase of HHs can result in two main drawbacks:

1. Likeliness: two problem instances that have similar values on their features are best solved with different heuristics.

2. Stagnation: the feature space becomes more reduced in variations as the search method explores the solution space.

It is posed that by using transformations, we can achieve better discrimination on a specific set of rules for a set of features, given that the features correctly represent a state of the problem, and also increase the variation of the feature between states because of an expansive effect that some transformations exert on the feature space.

Amaya et al. [3, 4] propose to use either explicit (e.g. directly applying a function to the features) or implicit (e.g. mapping the feature space to a higher dimensional space) transformations. For the scope of this research, only explicit transformations will be studied.

1.3 Hypothesis

It is believed that the generation of a HH for the current dispatching rules or low-level heuristics on the JSS problem could achieve better results than by using them in an isolated manner. Also, based in evidence on other domains, we believe that applying feature transformations could enhance the generated HHs and that the nature and extent of such enhancement can be defined by comparing them against HHs produced without transformations.

The following questions are to be considered for this investigation:

- How does low-level heuristics differ between each other, and to what extent is their behavior consistent throughout different instances of the JSSP?
- How can the current state of a JSSP solution be measured?
- What operations need to be carried out during the training process to generate HHs for the JSSP?
- How does the number of iterations selected for the training phase of the HHs affect their performance?
- How does the size of the instances selected for the training phase affect the training process?
- How scalable are the HHs when tested on larger instances than the ones used during the training phase?
- What has to be considered when defining the parameters used in the transformations of the features that characterize the problem state of the JSSP?
- Does applying feature transformations enhance the HH? What is the nature and extent of such enhancement? (in case it exists).

1.4 Objectives

The general objective of this investigation is synthesized as: “Design and implement a HH selection framework by SA search for the JSS domain and analyze the application of feature transformations to the generated HHs”.

Particular objectives are enlisted accordingly:

- Establish how does the solution given by a HH compare to the solution of the best heuristic for a given instance of the problem.

- Describe the solution space of the JSS problem in order to better adjust the abstraction structure used to model the problem.
- Test low-level heuristics and features on the solution of the JSSP.
- Generate HHs based on a SA algorithm, and experiment with the parameters of the SA algorithm to define how to adjust them.
- Present and analyze possible scenarios related to the computational time when generating HHs, in order to define the parameters and instances to be used on the experiments.
- Select two feature transformations from the literature to apply and analyze them in such a way that it is possible to report to what extent they enhance the solution of a HH generated without transformations.

1.5 Contributions

The research presented in this thesis contributes to the scientific community in different ways. The most important contributions are listed below:

1. **Applying a selection constructive HH approach with feature transformations to the JS domain**. To the best of the author's knowledge, there is no previous work on which selection constructive HHs with feature transformations have been used to solve the JSSP.
2. **The rationale behind the JS subproblem dependence that determines several key points to consider throughout the training phase of HHs for the JSSP**. One of the conclusions that were obtained throughout the development of this research is that JSS problems have a subproblem dependance inherent to them which provokes that constructive heuristics follow a similar path of construction of the solution. HHs overcome this drawback if the states of the problem are sufficiently well identified. Features are used to represent such states, and the main goal of the transformations is to help the training process so that HHs are able to distinguish the different problem states during the construction of the solution.
3. **An adaptive method to determine the most suitable parameters to apply the transformations, which could be extended to other domains of optimization problems**. To apply feature transformations, a range of the domain of each feature needs to be determined. This can be done empirically by studying how the feature values differ between the heuristics that are used to construct a solution. This process though can also be automated by maximizing the distances between the feature values across the solution path of a heuristic. A method for doing so is outlined and tested in this thesis. Results indicate that its application produces better and more stable HHs than by training without transformations or by applying them in an empirical way.
4. **A framework for studying HHs in the Job Shop domain**. This thesis is part of the previous work done by Amaya et al. [3]. HERMES is a framework that has been used

by that research group for studying selection constructive HHs. The original version of that framework is built in Java, was translated to Python, and a modular framework was built on top of it to be able to produce selection constructive HHs for the JSS problem, and also, a SA algorithm is introduced in the framework, with the possibility of adding future modules for other meta-heuristics.

5. **A proposal of future lines of work that derive from this research**. Since this is a first approach to solving JSS problems with selection constructive HHs, several ideas arose from this research, which could improve the results presented herein. An outline of those ideas and of several ways in which they could be implemented is described in the last chapter of this thesis.

1.6 Organization of the Document

The thesis is organized as follows. Chapter 2 presents a theoretical review of the concepts of scheduling, Job Shop scheduling, hyper-heuristics, and simulated annealing. Chapter 2 includes a review of the main classes of methods that have been used for solving the JSSP. Chapter 3 outlines the solution model that will be used in this research to solve the JSSP. Chapter 4 describes the methodology that is followed to determine if the proposed hypothesis stands and to meet with the objectives aforementioned. Chapter 5 presents exploratory experiments, both of isolated heuristics and HHs without transformations, which determine the line of work for the next stages of the methodology. Chapter 6 deals with the inherent effects of the instance size that impact the way a HH is trained. Chapter 7 introduces how the feature transformations are defined and tuned, and the experimental results achieved with this approach. Chapter 8 presents an overview of the research, its results and achievements, the extent to which the goals are met and possible research paths that could be explored in the future.

Chapter 2

Theoretical Framework

This chapter introduces the main concepts of scheduling necessary to situate the Job Shop Scheduling problem in that area of research. The main aspects that define Job Shop problems and what makes them different from other scheduling problems is also mentioned. Next, the definition of the Job Shop scheduling problem is introduced, by using the traditional notation found in the literature. Afterward, exact, approximation and artificial methods that have been used to solve the Job Shop problem are reviewed, with a focus in priority dispatching rules, which is a key topic in this research. Following this, the chapter defines hyper-heuristics and the different methods that have been used by researchers to produce them. Finally, details of the Simulated Annealing algorithm are presented, which is the meta-heuristic used in this thesis to train hyper-heuristics.

2.1 Scheduling

Scheduling is part of any process that involves time and resources. The design of a schedule comprises the allocation of activities or processes at a determined place and time, taking into consideration other activities involved and the available resources. The latter being either humans, machines, central processing units, means of transportation, etc. or any kind of resource to be consumed. The characteristics of the activities in a schedule range from just being defined for a specific period to having other implications such as consumption of other resources, complying with a sequence of priorities, having a due date, etc. Also, scheduling involves the definition of one or multiple objectives: maximization of resource allocation, minimization of total processing time, minimization of used resources, etc. The objective of a schedule depends on the context where the schedule has to be produced. Nevertheless, the goal is always that given the definition of what resource is the schedule expected to care of, find a schedule that optimizes that resource. Thus, scheduling is defined as "the allocation of scarce resources to tasks over time" [94].

Based on uncertainty in the information provided by an instance of the problem, scheduling is usually divided into two major categories [58]: deterministic and stochastic scheduling. The former involves problems in which no uncertainty is present (for the case of JSS, where job priorities, processing times and set-up times are known beforehand). The latter represents problems where there exists a probability distribution that represents the existence of a certain

characteristic of the problem. This research focuses on a deterministic model of scheduling.

2.1.1 Deterministic Models

There are several environments in which a deterministic schedule can be defined. Each has its own unique characteristics which differentiate itself from the others. The most notable domains in deterministic scheduling are described in this section. It is helpful to establish a reference framework and notation to describe them. The model proposed by Pinedo [95] was chosen to describe the Job Shop scheduling problem in this work. Denote n as the number of jobs and m as the number of machines. Let any job j_{ij} have the following characteristics (where i stands for the machine and j the job):

- Processing time (p_{ij}): the units of time required for the job j to be processed at machine i.
- Release date (r_j): the earliest time the job j can start in the schedule.
- Due date (d_j): the deadline for a job j to be finished. Usually, it is permitted for a job to finish after its due date, but a penalty is involved.
- Weight (w_j): a measure of how important is job j relative to the other jobs in the system.

With this notation, a number of machine environments are possible, being the most important ones [94]:

- Flow-Shop: for m machines in series, process each job on one of the m machines, and follow the same processing sequence on the machines.
- Flexible Flow-Shop: for c stages in series with a number of identical machines in parallel at each stage, process each job at a given sequence of stages. At each of those stages, the job is processed on one machine, but any will suffice.
- Job Shop: process each job j by following a specific sequence of machines different for each job. This case is thoroughly described in Section 2.2.
- Flexible Job Shop: analog to parallel machine environments for flow-shop but with a predefined sequence to be followed.
- Open-Shop: having m machines, each job has to be processed a given amount of times on a machine, in any order.

2.1.2 Stochastic Models

In a real-world shop, problems with machines, new jobs inexistent to the original problem, etc. can arise. Scheduling a shop with this uncertainties involves modeling this randomness [94]. The main difference when modeling this kind of shops is that p_{ij} are not crisp values of times, instead, distributions for processing times are used. Also, for the cases of stochastic models with release dates, a random variable is used. The main difference between the different models is how such distributions are formulated. When solving stochastic problems, the impact of the random variables must be taken into account.

2.1.3 Restrictions of Scheduling Models

Processing constraints also vary from one scheduling problem to another, since each model has its own set of restrictions. Examples of these constraints include: release dates, sequences setup times, no preemptions (if a job starts, it has to finish until it is completed), precedence constraints, machine breakdowns, restrictions on the eligibility of machines, a specific permutation to be followed throughout the system, no-wait policy, recirculation, etc. Also, the objective for an optimal schedule can vary from model to model. Generally, the objective is to minimize a function of the completion time C_{ij} of a job j on a machine i [95]. The specific definition of the objective function to be minimized (or maximized) will depend on the model.

2.1.4 Graham's Notation for Scheduling Problems

Throughout the years in research of the scheduling area, the (α, β, γ) theoretical notation introduced by Graham et al. [54] to denote any scheduling problem has been used as a standard. α describes the machine environment (single or multi-stage, etc.), β the job characteristics and constraints (due date, release date, etc.), and γ the objective function. With the use of this notation, various scheduling problems can be defined.

2.1.5 Static and Dynamic JSS Problems

Although scheduling problems are, in general, classified as stochastic or deterministic, for the JSS problem, it is important to note that another classification is by how possible jobs are released. This classification is divided into static and dynamic JSSP [1]. Whereas in static JSSP all machines and jobs are finite and known beforehand and available throughout the scheduling process, dynamic JSSP considers the case when jobs arrive continuously over the development of the schedule.

The definition of how the jobs arrive has varied depending on the research considered and has recently experienced a growing interest in the study of combinatorial optimization and computer science areas [12, 41, 102, 68, 118, 122].

Although dynamic JSS problems may be closer to real-world shops because of the consideration that jobs are not always predefined or known, the scope of this thesis is delimited to static JSS problems. The main reason for this is that, as said in the previous chapter, the current HH technique is similar to a model of HH that has previously been tested within static domains (such as CSP, bin-packing, Knapsack, among others) [3, 4, 52, 78]. Hence, the simpler nature of static JJS problems helps to establish a similar framework of reference to study how HHs will perform and also be able to study the effect of feature transformations.

2.2 The JSS Problem

Following Graham's notation [54], the JSSP is a $J_m||C_{max}$ problem with arbitrary processing times and no recirculation [95]. This is, a scheduling problem with n jobs to be processed on m machines with the objective to minimize the total time of the schedule, given by C_{max}, the completion time of the last job in the schedule. Many representations of this problem are

available in the literature [2, 7, 10, 95], but the disjunctive programming formulation is the one most often used [94]. The problem is mathematically formulated as follows.

Let N be all operations (i, j) and A the set of all sequence constraints $(i, j) \rightarrow (k, j)$ that require job j to be processed on machine i before it is processed on machine k. Let y_{ij} be the starting time of operatin (i, j), then the formulation is as follows:

$$
\begin{array}{ll}
\text{minimize } C_{max} \text{ subject to} & \\
y_{kj} - y_{ij} \geq p_{ij} & \text{for all} (i,j) \rightarrow (k,j) \in A \\
C_{max} - y_{ij} \geq p_{ij} & \text{for all} (i,j) \in N \\
y_{ij} - y_{il} \geq p_{il} \text{ or } y_{il} - y_{ij} \geq p_{ij} & \text{for all} (i,l) \text{and} (i,j), i = 1, ..., m \\
y_{ij} \geq 0 & \text{for all} (i,j) \in N
\end{array} \tag{2.1}
$$

The following constraints can be inferred from eq. (2.1): 1) no operation can start before another one with a higher precedence on a sequence constraint and 2) no two operations can be carried out at the same time in the same machine.

The static JSSP problem is known to be NP-Hard by reduction to the single machine scheduling problem [58, 73]. In fact, there are $n!^m$ possibilities to schedule the n number of jobs and m number of machines on a given shop floor [88]. A small example of the solution space for a JSSP of $m = 1$ is presented in Table 2.1.

Table 2.1: Example of the increasing complexity of JSS problem with n jobs and $m = 1$.

j	Possible schedules
2	2
4	24
8	40,320
16	2.09E23

2.3 Related Solution Methods

Solving a JSSP can be simplified to the task of constructing a schedule of activities. For $m = 1$ or $m = 2$, optimal polynomial time algorithms exist. An example for $m = 2$ with a fixed number of jobs is the one published by Brucker et al. [18]. The author shows that the JSSP with 2 machines can be reduced to the shortest path problem of an acyclic graph with $O(r^k)$ vertices, where r is the maximal number of operations per job and k the number of jobs. For $m > 2$, Garey et al. [48] provide a proof of NP-Completeness. This has resulted in several methods for solving this problem. Traditionally, there are three main classes of models for solving a JSSP [24, 38, 96]:

1. Using exact methods for combinatorial optimization.
2. Using approximation methods, such as dispatching rules.
3. Using artificial intelligence methods.

2.3.1 Exact Methods

Exact methods rely on a direct mathematical approach such as analytical and deterministic ones. The following lines present an overview of such approaches.

Integer and Mixed-Integer Programming Methods

Some of the first approaches for solving scheduling problems include the works of Bowman [16] and Kondili [66], although slightly different in the specific model, both consist of a linear-programming method with a set of constraints and objective functions where variables are indexed based in time. Also, a similar integer-programming model was published by Wagner and Harvey [111], but instead of indexing variables based in time, they do it by optimizing the variables based in a rank parameter which involves the machine and job of each activity. Another formulation is based on a disjunctive graph [80]. Such formulation has been tackled both with integer programming [8, 9] and with mixed integer programming methods (MIP) [10].

More recently, Ku et al. [67] conducted an empirical study of the computational efficiency for the methods aforementioned. Both time and rank resulted to be less computationally efficient than the disjunctive graph one. Because of this, they use three different kinds of MIP optimizers to test the disjunctive model. The experiments were carried out both in a single thread and in an 8-thread parallel configuration, for several problem instances of sizes between 3×3 (3 jobs and 3 machines) and 20×15. Average computational time for finding the optimal solution for each instance was reported for a set of 10 instances for each size. For instances of size 20×15, the methods could not reach optimality within a timeframe of 3600 seconds. For illustration purposes, results for instances of size 3×3 and 15×15, both for single and multi-threaded versions, are shown in Table 2.2 . The 10 15×15 instances correspond to the benchmarks published by Taillard [107]. Note that optimality was reached for half of the instances of size 15×15 when the optimizer is single-threaded. Although optimality is feasible with optimizers for MIP formulations, and available computational power is increasing, the inherent NP-Completeness of the JSS problems makes it very difficult to reach optimal solutions in a reasonable timeframe for larger instances.

Table 2.2: Comparison of average times for finding optimal solutions to 3×3 and 15×15 instances in single and 8-threaded MIP optimizers, as reported by Ku et al. [67]. Each set contains 10 instances, reported time corresponds to average time for one instance. *n.o.s* stands for the number of optimal solutions achieved in the set.

Size	**1-thread (time / n.o.s)**	**8-threads (time / n.o.s)**
3×3	0.01s / 10	0.02s / 10
15×15	2272.75s / 5	1585.79s / 8

Constraint Programming Methods

Constraint programming (CP) involves expressing the relations between the variables of an optimization formulation are stated in terms of constraints. For the JSSP, the disjunctive model can be restated as a CP one and several CP methods have been proposed under this approach [11, 109]. On the computational efficiency study mentioned for MIP solvers [67], an experiment was conducted using a CP formulation, results show that for the 15×15 instances, on average, the solver takes 48s to reach an optimal solution for all the tested instances. Besides this, for the case of instances of size 20×15, optimality was reached for 6/10 instances, with an average computational time of 1924.35s for each instance. CP is currently being studied by a number of researchers, hereby only a small illustration is presented. Besides the fact that very good results were achieved for a larger instance size than MIP, an increase of 5 jobs produces a time increase of over 40%. The state of the art method for solving CP JSS problems is the work of by Beck et al. [13], and involves combining CP with a tabu search strategy.

Lagrangian Relaxation and Decomposition methods

Another approach for finding solutions to the mathematical formulations previously presented has been the use of Lagrangian non-negative multipliers to relax constraints [60] such as precedence. Also, several decomposition techniques have been proposed to overcome the difficulties posed by MIP formulations [96]. Essentially they work by separating the JSSP into subproblems, and then try to solve each subproblem (whose optimal solution is easier to find) and in some particular way build the complete schedule. Although sub-optimal solutions are found by this kind of methods, they are presented in this section because of the nature of the methodology followed, where a particular problem starts from the disjunctive formulation and then successive decomposition or relaxation techniques are used. Examples of these approaches include the works by Fisher [42, 43]. The main contribution of this author is that he uses Lagrangian multipliers in subproblems to obtain optimal solutions, and also proposes lower bounds on the cost for the further application of the branch-and-bound technique (described in the next subsection). Also to note is the work by Chen et al. [28] in which they relax precedence with Lagrangian multipliers and also apply a decomposition technique.

Branch-and-bound method

The branch-and-bound method relies on structuring the solution to the problem as if it were a decision tree. Each node of the decision tree corresponds to a step of the solution, each branch of a node corresponds to a possible path to take, until leaf nodes are reached, which correspond to the solution of the scheduling problem [87]. Hence, the algorithm consists of searching the decision tree for a solution. Since it is impractical (and computationally speaking intractable) to search the whole tree, different strategies are used to reduce the search. Many examples of researches that use this technique have been published [6, 25, 82, 84], the difference between the proposed methods mainly relies on the way the algorithms produce an estimation of a lower bound for a solution and a calculated best upper bound which guides the search and in the way the methods prune the decision tree to speed-up the search strategy. Improvements have been made on branch-and-bound methods, but more because of the computational power

available than for the improvement of the techniques by themselves. Also, scaling remains as an important bottleneck in this kind of solution methods [60].

2.3.2 Approximation Methods

Two main methods are frequently described in the literature: priority dispatching rules and the shifting bottleneck procedure.

Priority Dispatching Rules

Priority dispatching rules serve as simple instructions to select the next activity to be scheduled; because of this, they have been present in JSS since the first attempts to systematically solve these problems were made [95]. Also, they are sometimes referred with a different name, having the same meaning (dispatching rule, scheduling rule, sequencing rule, or heuristic) [96], but some authors such as Gere [50] do not agree with this discrepancy in the names, and tries to define what are the inherent characteristics of each concept. In his definition of heuristics, he considers that they are rules of thumb, and when they are combined with other rules then a scheduling rule is formed. Several authors have presented surveys of well-known dispatching rules. Panwalker and Iskandar [91] present a survey of more than 100 rules. In this paper, the authors make a distinction between priority rules (no matter if priorities are as is, result from the combination of several rules or with a weighted index), heuristic rules (which the authors say that "involve a more complex consideration", that is, rules that derive from an appreciation of other information) , and other (for example, rules defined only for a specific shop). Another popular survey is the one published by Blackstone, et al. [55]. The author makes the point clear that there is no specific best rule for any circumstance, but studies a number of applications and discusses why such and such heuristic is better. Haupt [57] does a similar work than the other authors already mentioned, and many of the priority rules correspond between the authors. Haupt states that the shortest processing time (SPT) is among the most used, commented and efficient dispatching rules. Other trends involving dispatching rules are evolving combinations of them to find new ones, in the work by Tay and Ho [108] usage of genetic programming serves to generate new composite dispatching rules, and the work by Jun et al. [61], where the authors learn new dispatching rules for a flow shop scheduling problem by using a random forest technique. The experiments reported by the authors show that the generated heuristics outperform the original ones.

Shifting Bottleneck Procedure

The Shifting bottleneck procedure (SB) relies on some ideas taken from the branch-and-bound method proposed by Carlier [25]. It was first published by Adams et al. [2]. The general procedure involves iteratively separating the original JSSP into subproblems comprised of only one machine. Each one-machine problem is solved to optimality by using branch-and-bound. The machine that ends in the latest time becomes the bottleneck, which is then solved by taking into consideration any previously scheduled machines. Also, every time a bottleneck machine is solved, the sequence of previously solved machines is reoptimized by solving the one-machine relaxation. A comprehensive study on results for several variations of the SB

procedure tested on different instances of the JSSP was published by Demirkol et al. [35]. They compare the results against five different priority dispatching rules and show that SB methods consistently perform better than them. Adams [2] discussed several problems that could arise from using SB, such as a deadlock cycle that occurs in some instances. More recently, Wequi and Aihua [116] investigated the deadlock effect and proved a theorem which allowed them to design and implement an improved version of the original SB (ISB), which is more efficient in terms of computational time and also generates better solutions for most of the tested instances.

2.3.3 Artificial Intelligence Methods

Artificial Intelligence (AI) relies on a several techniques which help an algorithm to be trained over a set of instances and then be able to apply such knowledge over a previously unseen instance. Some of the most important AI methods are described below.

Tabu Search

Several variations on Tabu Search (TS) methods for JSSP have been tested. In essence, TS searches a solution space while maintaining a list of previously explored solutions and the moves that produced them, and while performing the search, moves that correspond to bad solutions are not allowed. One of the most notable contributions is the one described by Nowicki and Smutnicki [89] called TSAB. It remains as one of the best TS applications to JSSP. The main characteristic of the approach is that they use a neighborhood of solution comprised of blocks that represent part of a critical path. TSAB found the optimal solution to a JSSP instance which had not been solved of size 100×100 in a few seconds. And also in instances with about 2000 operations, results show that TSAB is able to produce solutions that deviate around 4-5 % from the optimal [71]. More recently, in the works of Zhang, Li, Guan and Rao [120] TS has also been used. Also, Peng, Lü and Cheng [92] use a Tabu Search/Path Relinking (TS/PR) strategy, which resulted in the improvement of many upper bounds of instances from the benchmark set published by Storer et al. [106], and the optimal solution of instance SWV15 of the set, which according to the authors, had not been solved for 20 years.

Genetic Algorithms

A number of different configurations and representations for the JSSP has been used when applying Genetic Algorithms (GAs). Cheng et al. [29] reported 9 categories of configurations, based on this survey, one can see that some of them involve encoding the schedule as chromosomes, and directly use the genetic functions and operators for new offsprings. Also, other approaches have been to evolve a list of priority rules, and then apply it to generate the solution of the problem. GA methods can also differ between each other not only in the representation of the JS model but also in the variable defined with an objective function that the algorithm seeks to optimize. Some of the earliest evidence of using GAs for JSS include the work by Davis [34] and Liepins et al. [76]. Wu et al. [117] published a comparison study between several GA strategies for the JSSP and traditional local search techniques. In

general, GAs tend to be more stable than the search metaheuristics and also achieve better results in terms of the latest ending activity of the schedule (commonly known as makespan). Cheung and Zhou [31] present a genetic algorithm that works in a JSSP with setup times for each activity. The algorithm has two phases: 1) the construction of schedules by using one of two rules, the shortest processing time dispatching rule or the most work remaining dispatching rule, adapted for the consideration of setup-times; 2) a GA approach on the generated schedules. The algorithm had outstanding results when compared to previous ones, and although it is applied on JSSP with setup-times, the same kind of model could be adapted for classic JSS problems. The proposed hybrid GA strategy achieved better results than the ones produced by the used heuristics alone when exerted over instances of sizes 10×10 and 20×20. Approaches, where GAs are combined with search metaheuristics, are also present in the literature [113, 53, 45, 114]. Of more recent developments, Wang et al. [112] published an adaptive GA for solving classical JSS problems where the objective is to minimize the makespan. One of the main features of the algorithm is that it has a faster convergence rate than other similar approaches. Four main characteristics distinguish this method from the others: multi-population, adaptive crossover probability, adaptive mutation probability, and the elite replacing mechanism. They compare the approach against Simulated Annealing, TS, the new island GA model proposed by Kurdi et al. [69], and the SA-GA hybrid model [113], with very similar results.

Neural Networks

Several recent studies show evidence of using neural networks (NN) to solve the JSSP. A novel back-error propagation method is suggested by Jain et al. [59], the JSS problem is encoded in a way that the network grows linearly with the scale of the problem. This helps the network to perform better than other methods. Despite this fact, Jain et al. [60] does a survey on most of the available NN at the time of the research and shows that most of the proposed methods have to be joined with other hybrid techniques such as GAs to perform optimization. The survey is consistent with what was found during this literature review, that there are not many methods in the literature that use NN to optimize JSS problems. Nevertheless, an interesting example was found, because of its similarity to the solution model proposed in this thesis. El-Bouri and Shah [40] present a NN that selects from a set of four dispatching rules (shortest processing time, longest processing time, least work remaining, most work remaining, and processing time + work-in-next-queue) based in three measures of the problem instance, which include: normalized total processing time at each machine, normalized variance of processing times for each machine, and normalized mean routing order for each machine. The three parameters serve as input for the NN and the output is the chosen heuristic for the problem. The training set was comprised of a balanced mix of 10×10, 15×15 and 20×20 instances. The network started with 1,500 instances, with increases of 500, until 7,500 instances. Trained NNs where tested in 10 training sets, where instance size ranges from 10-100 jobs. Results are reported as comparisons of total makespan for each set, hence, it is difficult to compare this information to other methods besides the ones presented in the research.

2.4 Hyper-heuristics

The term was first used in 2000 to describe special "heuristics to choose heuristics" [32]. Another more recent survey that includes a classification for different approaches was published by Burke et al. [20], which also proposes as a definition that a "hyper-heuristic is an automated methodology for selecting or generating heuristics to solve hard computational problems". Many classifications exist for HHs:

- *Constructive vs. local search methods* [7]. Local search hyper-heuristics start from a complete initial solution and select appropriate heuristics to lead the search in a promising direction. Constructive hyper-heuristics build a solution incrementally by selecting heuristics at different stages of the problem.

- *4 level classification* [26, 20]. (i) HH based on the random choice of low-level heuristics, (ii) greedy and peckish hyper-heuristics, which requires preliminary evaluation of all or a subset of the heuristics in order to select the best performing one, (iii) metaheuristics based hyper-heuristics, and (iv) HHs employing learning mechanisms to manage low level heuristics.

- *Selection vs. Generation* [20], where selection HHs refer to methods for choosing heuristics and generative HHs methodologies to generate heuristics. Also, regarding learning, this same classification proposes to distinguish between online and offline learning. Being the former a HH that learns while solving the problem and the latter to first train the method with several instances and then prove that it works on a set of unseen test instances.

2.4.1 Applications

Hyper-heuristics have been used by researchers to solve a broad range of problems. Some of the main exponents of the usage of hyper-heuristics by the scientific community are Pillay et al. [93] and Burke et al. [21]. Some of the problems and the main works that include the use of hyper-heuristics are hereby outlined.

There are several domains for which constructive hyper-heuristics have been used. For the bin-packing domain, some examples include the works of Ross et al. [99] where they use an evolutionary algorithm to produce selection constructive hyper-heuristics which apply a different heuristic at each state of the solution process. Another example for this domain is a generative approach, proposed by Sim and Hart [104]. They use a single-node genetic programming algorithm to generate constructive heuristics for the 1-D bin-packing problem. Another frequent domain is timetabling. Examples include the works of Burke et al. [23], where tabu search is used to permutate a graph of the heuristics used to produce timetables for two study cases: exam and course timetabling problems. Moreover, the work of Rodríguez et al. [97] is an illustrative example done on the flow shop domain. The authors generate permutations of the dispatching rules with a Genetic Algorithm.

For the case of perturbation low-level heuristics, a particular example belonging to the domain of packing is the work by Dowsland et al. [37]. The authors tackle the problem of selecting shipper dimensions for the optimization of volume utilization. They take the ideas

of a previous work of Burke et al. [22] similar to the work also described in the previous paragraph [23], but instead of using tabu search, they combine that approach with a simulated annealing framework. Another notable example of perturbative approaches reviewed by Burke et al. [21] is the work by Ropke et al. [98], where the authors study the pickup and delivery problem for constructing routes. In this case, the hyper-heuristic involves an adaptive method for choosing heuristics based on historical performance, and furtherly updating each of the heuristics' performance during the solution process.

2.4.2 Other High-level Methods

Most of the approaches used to produce hyper-heuristics involve the use of a search strategy that relies on either in genetic algorithms or a variation of them, or other search-based meta-heuristics such as simulated annealing and tabu search. However, recent trends [93, 21] include the exploration of other strategies. For example, Sabar et al. [100] construct a tree of low-level heuristics and perform Monte Carlo tree search to identify the sequences of heuristics to be applied. Another one is the work by Kheiri and Keedwell [64]. In this case, instead of using a search strategy, they produce a Markov model to assert the performance of sequences of heuristics to solve the optimization problems. Although this is not a search strategy by itself, it illustrates how other trends of research in the area of hyper-heuristics have arisen.

2.5 Simulated Annealing

In the proposed solution model, the HH is trained with a Simulated Annealing process. This metaheuristic, originally presented by Kirkpatrick, Gelatt and Vecchi [65], is one of the simplest and most general techniques which turned out to be one of the most efficient ones. The algorithm mimics the crystallization process during cooling or annealing: when the material is hot, the particles have high kinetic energy and move more or less randomly regardless of their and the other particles' positions. The cooler the material gets, however, the more the particles are "torn" towards the direction that minimizes the energy balance. The SA algorithm does the same when searching for the optimal values for the decision parameters: it repeatedly suggests random modifications to the current solution, but progressively keeps only those that improve the current situation. SA applies a probabilistic rule to decide whether the new solution replaces the current one or not. This rule considers the change in the objective function (measuring the improvement/impairment) and an equivalent to "temperature" (reflecting the progress in the iterations). Dueck and Scheuer [39] produced a variation of the algorithm by suggesting a deterministic acceptance rule instead which makes the algorithm even simpler: accept any random modification unless the resulting impairment exceeds a certain threshold; this threshold is lowered over the iterations. This variation is known as Threshold Accepting (TA). The pseudocode of the complete SA algorithm is due to Maringer [81] and shown in Algorithm 1. There are several parameters of the SA algorithm that can be adjusted. A notable one is the cooling schedule, that is, the procedure by which the temperature is updated. The most used is the traditional one defined by Kirkpatrick et al. [65] which is shown in Equation 2.2. However there are other approaches in the literature [79], such as a logarithmic cooling

schedule by Geman and Geman [49] shown in Equation 2.3 but it has been shown that is slow in terms of convergence [79]. Other cooling schedules are available in the literature, a formal review of these and also of adaptive cooling schedules is described in the work of Zerubia et al. [90].

$$T_k = T_{max} * e^{\frac{\alpha * k}{N}} \text{ for } k = 1, ..., N \text{ and } \alpha = -\log(T_{max}/T_{min}) \tag{2.2}$$

$$T_k = \frac{c}{\log(1+k)} \text{ for } k = 1, ..., N \text{ and } c \text{ a constant (usually 1)} \tag{2.3}$$

Algorithm 1 Simulated Annealing

```
1: Generate a feasible solution X
2: Let E be the energy of a solution
3: Let P be the acceptance probability
4: Let T be the temperature of the system
5: T ← 100
6: while A halting criterion is not met do
7:     Generate new solution X' by randomly modifying X
8:     if E(X') ≥ E(X) or P(E(X'), E(X), T) ≥ random(0, 1) then
9:         X ← X'
10:    end if
11:    Update T according to the cooling schedule
12: end while
```

SA has been broadly applied to many problems, mainly to those belonging to the combinatorial optimization domain. Recent examples of the use of SA include the work of Karagul et al. [63] where they employ a SA to optimize solutions for the Green Vehicle Routing Problem with Fuel Consumption. Also, Wei et al. [115] use a SA for the capacitated vehicle routing problem with two-dimensional loading constraints. Other works have been focused on studying how different variants of SA algorithms can be used for a problem domain. One example of such is the work by Ghaffarinasab et al. [51] for solving Hub location problems, which are a class of problems which contain transportation, postal services, telecommunications, among other logistic related problems. Also, a notable example is that of Franzin et al. [44]. The authors describe and implement a framework that is automatically configurable based on several variants of SA algorithms on which several combinatorial optimization problems can be tested. They also study the impact of each component and parameter of SA in several use cases.

2.6 Feature Transformations

The idea of using feature transformations to enhance hyper-heuristics is relatively new in the literature. Nevertheless, the approach followed has been widely implemented in other computer science disciplines such as data mining [47], where several steps need to be applied to pre-process data. In general, they can be summarized into three steps [4]: selection, generation, and transformation. It is this latter step what is deeply related to the research proposed

in this thesis. Feature transformations are used to reshape the feature values in order to aid data mining algorithms during the training phase. An illustrative example is in the work by Yu et al. [119]. The authors study the recognition of persons in photos that are provided from several camera views. In this case, they use feature transformations for each camera view to account for feature distortions that arise from each view. Although very far related to the problem that is posed in this thesis, the idea is the same. To aid the training process, and account for variations in feature values among the different instances. Specifically for hyper-heuristics, feature preprocessing has been an approach used in several works [105, 86, 56], but all of them focus preprocessing in feature selection or generation [4]. In this matter, after thorough research of the literature, it was found that Amaya et al. [3, 4] are the only authors to use feature transformations for enhancing hyper-heuristics. The authors apply several kinds of feature transformations to feature values of Constraint Satisfaction Problems and perform confirmatory experiments on the Knapsack domain. The hyper-heuristics produced by the authors are selection constructive, evolved with a genetic algorithm. The authors tested three transformations: linear, S-shaped and kernel-based. Results indicate that for the case where transformations were applied considering only two features, the generated hyper-heuristics perform in average as good as the best heuristic, and when increasing the number of features, the hyper-heuristics generated with the S-shaped transformation did not perform as well as the ones generated with the use of the kernel.

2.7 Summary

Scheduling is a difficult combinatorial problem that has been studied by a broad number of authors. Several models of schedules are present in the literature and can be divided into deterministic and stochastic. A kind of deterministic model is the Job Shop Scheduling Problems. It is defined as a $J_m||C_{max}$ problem with arbitrary processing times and no recirculation. The Job Shop problem has been solved in the literature by three kinds of methods. Exact methods include Integer and Mixed-Integer programming methods, and constraint programming methods. This type of methods produce optimal schedules, but are very intensive in computational terms and can take too long to produce an optimal solution. Several techniques such as branch-and-bound have been used to relax the problem and aim to produce sub-optimal solutions. Other kinds of methods are approximated ones. These include the shifting bottleneck procedure and the use of priority dispatching rules. Both of them have been studied thoroughly in the literature. Next are artificial intelligence methods, with Tabu search being the one that has shown to be of best performance. Also, hyper-heuristics have been used in the literature to solve combinatorial optimization problems. There are several types of hyper-heuristics. They can be divided into four main categories, depending on the way they are formulated and the action they exert onto the problem. Only a single example of selection constructive hyper-heuristic for the Job Shop domain was found. Training hyper-heuristics involves the use of a meta-heuristic or another search method. Several methods have been used in the literature, one of them being simulated annealing, which has already been applied to other operations research problems, with promising results. Finally, there is evidence that using feature transformations to train selection constructive hyper-heuristics on other domains has proven fruitful in terms of enhancing the training process.

Chapter 3

Solution model

The main subject of this research is to solve a JSSP by using HHs and try to enhance their performance by using feature transformations. The following sections describe how a HH is to be represented, the features used to characterize a state of the JSSP, and the selected heuristics for this research. Also, the process by which the HHs are produced and trained with the SA algorithm is described. And the selected JSSP instances to test HHs on, how were generated and what are their main characteristics. Finally, the definition of each of the transformations to be used is also described.

3.1 HH Formulation

The HH model proposed in this work is taken from the works previously developed by Ross et al. [99] for solving bin-packing problems, with some modifications. This block model has also been used by several other authors such as Lopez et al. [78] and Amaya et al. [3]. The model is defined as an array where an action or rule r_i has features f_{i1}, f_{i2}, ..., f_{iq} which call to such action. These features-action sets are called *blocks*. An example of such representation is shown in Figure 3.1. The HH works by calculating each feature of a state S_t of the problem and then comparing it to each block. This comparison process is made by taking the Euclidean distance of the state from S_t to each block and returning the action of the block with the smallest distance.

Note that the number of features for each rule can be different among the rules, nor each

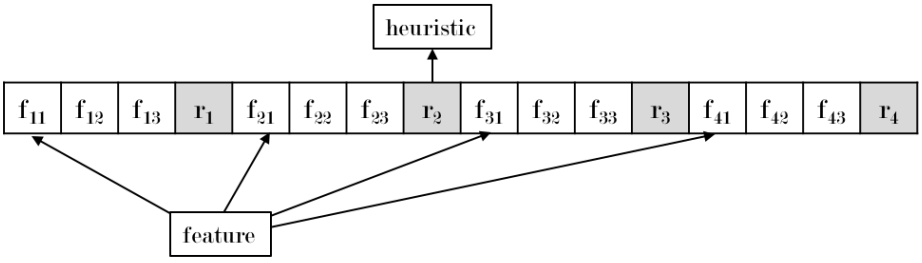

Figure 3.1: Example of a HH with 4 blocks. Features f_{iq} correspond to specific actions r_i, the latter representing a call to a specific heuristic, in this case, rules 1-4.

rule unique in terms of the heuristic it represents, but for the purposes of this research, a fixed number of features in each rule will be used, which corresponds to the number of features available. Nevertheless, the number of blocks can be larger than the number of available heuristics.

3.2 Problem Characterization

Two kinds of features are going to be used to describe the state of the JSSP. The chosen features were selected based on those found in the literature [83, 85] and by doing empirical studies to verify that they changed throughout the solution process and that produced values bounded to [0,1]. Mirshekarian et al. [85] did an extensive study of 380 features for the JSSP. The authors study the statistical relationships between the features and the optimal makespan for 15,000 JSS instances. It is interesting to see that one of the main conclusions of that research is that it highlights the importance of trial-and-error, and that although individually some features were better than others, when considered as groups in terms of the data they are calculated on, no group outranked another. Then, the authors emphasize the importance of selecting features based on the way they perform for a specific set of instances. Hence, a further study could be done to determine the extent to which the selected features in this thesis were appropriate, by taking the conclusions of this research on how they behave on the JSS instances used.

Two classes of features can be distinguished from the set. Features can either describe the state of the schedule, while others provide information related to the state of the solution. Although no formal feature selection methodology was followed to choose a proper set, the intuitive idea of choosing features of both classes was followed. With these considerations, the following features were selected:

Features that describe the state of the schedule (related to the solution)

1. Average processed times (APT): the ratio between the sum of processing times of already processed activities and the sum of processing times of the whole list of activities. This feature gives a rough idea of how advanced the state of the process is, and although it includes information about the current state of the schedule, it -at the same time- provides information about the activities to be scheduled.

2. Dispersion of processing time index for scheduled activities (DPT): for scheduled activities, the ratio between the standard deviation of processing times and the mean of processing times.

3. Percentage of slack in makespan (SLACK): the ratio between the amount of unused machine time (slack) in the whole schedule and the current make-span of the schedule. The more slack, the less compact the activities are, but also the more space that could be used by smaller activities.

Features that describe the state of pending activities (related to the problem)

4. Dispersion of processing time index for pending activities (DNPT): for pending activities, the ratio between the standard deviation of processing times and the mean of processing times.

5. Average not processed times (NAPT): the ratio between the sum of processing times of pending activities and the sum of processing times of the whole list of activities. This is exactly the complement of APT.

6. Average pending processing times per job (NJT): for all pending jobs, sum the number of processing times normalized for each job. Then, divide such amount by the number of pending jobs.

3.3 Decision Rules (Heuristics)

The heuristics selected for this research were taken from the references described in Section 2.3.2. For the purposes of this research, the terms "heuristic" and "low-level heuristic" refer to the same concept: A rule that specifies which activity to schedule next. It is intractable to test all existing heuristics, and also, since this research relies on a technique that has not been applied yet for this domain, it is convenient to maintain a tight set of heuristics. The set of heuristics chosen were tested empirically to assert if they were able to produce dissimilar results, and also to produce a set of not only heuristics that have been found to perform well (such as EST and MRT), but also include others that are not that good. Experiments related to this phase are presented in Chapter 4. With these considerations, each heuristic chosen for this research is described below.

Let U_a be the list of activities to be scheduled. Let $S_i = (a_{j,i}, t_{a_j})$ be a list of tuples where i stands for the machine number, $a_{j,i}$ is an activity of job j that goes to machine i, and t_{a_j} is the time where activity a is being scheduled in job j. Said heuristics are the following:

1. Shortest Processing Time (SPT): from U_a select the activity $a_{j,i}$ with the shortest p_{ij}.

2. Longest Processing Time (LPT): from U_a select the activity $a_{j,i}$ with the longest p_{ij}.

3. Maximum Job Remaining Time (MRT): from U_a select the job that needs the most time for it to finish. Return the first possible activity (in precedence order) that corresponds to said job in the first available time.

4. Most Loaded Machine (MLM): in U_a, find the machine i which has maximum total processing time. The heuristic will return the activity a_j that has the lowest possible t_{a_j} if scheduled in machine i. If no activity is possible, then for the sets of machines minus machine i, select the next machine with maximum total processing time until a suitable activity is found.

5. Least Loaded Machine (LLM): in U_a, find the machine i which has minimum total processing time. The heuristic will return the activity a_j that has the lowest possible t_{a_j} if scheduled in machine i. If no activity is possible, then for the sets of machines minus

machine i, select the next machine with minimum total processing time until a suitable activity is found.

6. Earliest Start Time (EST): for U_a, get the activities which can be scheduled at a given state (possible activities). Find the job that has the earliest possible starting time at the problem's current state, and select the activity that corresponds to the said job from the list of possible activities.

3.4 SA Method

To train a suitable HH for solving the JSSP, both the number of features for each block and the parameters for each rule have to be determined. As stated in Chapter 2, many techniques are available in order to achieve such representation. Some previous works on hyper-heuristics [19, 3] have used tabu or evolutionary algorithms to search for a good model of hyper-heuristic. Despite this fact, it was decided to employ the simulated annealing (SA) algorithm [65] with the traditional exponential cooling schedule (Equation 2.2) in order to construct the HH. There are reasons for exploring this search method besides other possible approaches. First, there is evidence that for several domains, using a simulated annealing approach for selection perturbative HHs has been successful [14, 7, 62], and while other selection constructive hyper-heuristics have been widely trained with genetic algorithms and tabu search, it is interesting to explore another meta-heuristic. Second, SA has a stochastic acceptance criterion. This characteristic has been shown to be promising for selection hyper-heuristics that work on heuristics of logistic related problems [33].

The parameters of the SA and basic operations to construct each candidate HH that the algorithm can use have to be defined. The output of the algorithm is the hyper-heuristic that gets the best fitness function in the search process. This method will run for all the pre-defined training instances of the problem, each time the HH will be tested on all the training instances, and its fitness will be evaluated according to the objective function which is described in Section 3.4.1. The pseudo-code to train a HH is described in Algorithm 2. Note that when calculating the fitness of a HH (line 11), the procedure described in Section 3.4.1 has to be followed. Also, note that when generating a neighbor HH (line 6 in Algorithm 2) there are three possible modifications to the current HH (observed in Figure 3.2), being:

1. Randomize feature: A random block and feature are selected, whose value is replaced for a new random value.

2. Add new block: The size of the HH is increased by 1 block, giving it a new action with random values for its features.

3. Remove block: The size of the HH is decreased by 1 block, eliminating a randomly selected block of the HH. This randomization option is omitted if the current HH has only one block.

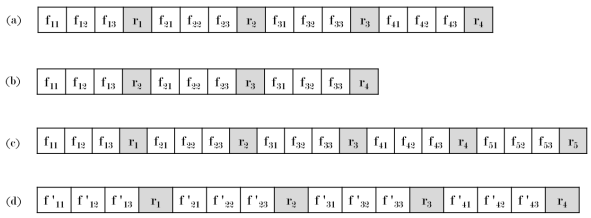

Figure 3.2: Possible operations done by the SA over a candidate solution (a): remove block (b), add block (c), randomize the feature of a block (d).

Algorithm 2 Simulated annealing implemented for HH training.

1: Set initial temperature t to a defined value
2: Set cooling rate c to a defined value
3: Generate an initial hyper-heuristic H
4: Set the best hyper-heuristic $H* = H$
5: **while** The system is not cool, meaning $t >$ threshold **do**
6: Create a neighbor hyper-heuristic H_n by randomly modifying H
7: Calculate fitness function for H and for H_n
8: **if** Acceptance probability $(H, H_n) >$ random number **then**
9: $H \leftarrow H_n$
10: **end if**
11: **if** fitness of $H <$ fitness of $H*$ **then**
12: $H* = H$
13: **end if**
14: Update temperature $t = t(1 - c)$
15: **end while**
16: **return** H^*.

3.4.1 Objective Function

To measure how good is a JSS solution, a comparison between the final makespan of the schedule against a reference has to be made. Also, the SA algorithm optimizes an objective function d as a reference to the best single heuristic for that same instance. This can be seen as a deviation of the makespan achieved by the HH from the makespan the best heuristic was able to produce for the same instance:

$$d = \frac{C_{s_i} - C_{b_i}}{C_{b_i}}, \tag{3.1}$$

C_{s_i} is the make-span achieved by the HH on the instance i and C_{b_i} is the best make-span achieved on instance i by any of the heuristics. The HH will be trained with the SA algorithm for several instances, thus, fitness results for all instances are accumulated with a general objective function used in the SA algorithm, which is:

$$d(HH) = \frac{\sum_{i=1}^{n} \frac{C_{s_i}^{HH} - C_{b_i}}{C_{b_i}}}{n}, \tag{3.2}$$

where HH is the corresponding HH and n is the number of instances that each candidate hyper-heuristic tries to solve.

3.5 Instances

The instances of JSS problems publicly available by Taillard [107] are used for testing purposes. These are among the most frequent in job shop research articles, and although are not the only ones, seem to be suitable for a first approach to the model proposed in this thesis. The author generated a number of JSS problems whose size was greater than that of the examples published up to that date (1993). Also, he published a set of 260 instances of different sizes which -according to the author- where the most difficult ones to solve by a number of methods. The instances generated by Taillard have the following characteristics: fixed processing times, no set-up times, no due dates nor release dates, and a lower -theoretical- bound and an upper bound (best one achieved so far). Also, the times for each activity are bounded between 1-99, hence, produce schedules that are easily managed in memory.

Table 3.1: Example of a JSSP instance of size 3×3.

	Machines			**Times**		
job 1	1	2	3	8	25	12
job 2	2	3	1	43	98	1
job 3	3	1	2	22	12	32

The author also provides a method to estimate a lower bound for each instance [107]. He concludes that an approximation of the lower bound for an instance can be calculated with the following formula:

$$lb = \max\left(\sum_{j=1}^{n} p_{ij} \forall i, \sum_{i=1}^{m} p_{ij} \forall j\right) \tag{3.3}$$

That is, the maximum amount of time required by a job or a machine.

Also, several instances will be used to test the trained HHs and make confirmatory experiments. Lawrence [72] produces 40 instances of various sizes. 5 instances of size 15×15 are used (la36, la37, la38, la39, la40). Demirkol et al. [36] produced 80 instances of different sizes, from this set, 10 instances of sizes 20×15 were selected (dmu01-05 and dmu41-45).

3.5.1 Instance Generator

Taillard's [107] instance generator was used to produce instances for the purposes of this research. The pseudocode is described in Algorithm 3. For a given number of jobs and machines, the algorithm will generate the information necessary to describe the JSSP. This is, for each job, the processing time of each activity and the sequence of machines the job has to follow. It works by generating two matrices: one for processing times of each of the job's activities, and the other one for sequences of machines per job. The processing times are generated by uniformly selecting times from the interval [1,99]. The random number

generator is described in [107] but to produce a uniform distribution, the author takes the ideas from the generator previously published by Bratley et al. [17]. The pseudocode for this generator is described in Algorithm 4. Note that constants a, b, c, m in this algorithm are specific for the generator to return a uniform distribution.

ALGORITHM 3
Taillard's instance generator.

Let m be the number of machines and n the number of jobs.
$T \leftarrow n \times m$ array which represents the processing times of the j_{th} operation of job i.
$M \leftarrow n \times m$ array which represents the machine in which an activity i, j it to be processed.
$t_{seed} \leftarrow$ seed for times.
$m_{seed} \leftarrow$ seed for machines.
for $i = 1$ to n **do**
    for $j = 1$ to m **do**
        $t_{ij} \leftarrow floor(99 * \text{UNIF}((t_{seed}))$
    end for
end for
for $i = 1$ to n **do**
    for $j = 1$ to m **do**
        $ma_{ij} \leftarrow j$
    end for
end for
for $i = 1$ to n **do**
    for $j = 1$ to m **do**
        Swap $ma[i, j]$ and $ma[i, j + floor((m - j + 1) * \text{UNIF}(m_{seed}))]$
    end for
end for

ALGORITHM 4
Uniform random number generator.

function UNIF(x)
    $a \leftarrow 16807, b \leftarrow 127773, c \leftarrow 2836, m \leftarrow 2^{31} - 1$
    $kl \leftarrow x/b$
    $x \leftarrow a * (x \bmod b) - kl * c$
    if $x < 0$ **then**
        $x \leftarrow x + m$
    end if
    return x/m
end function

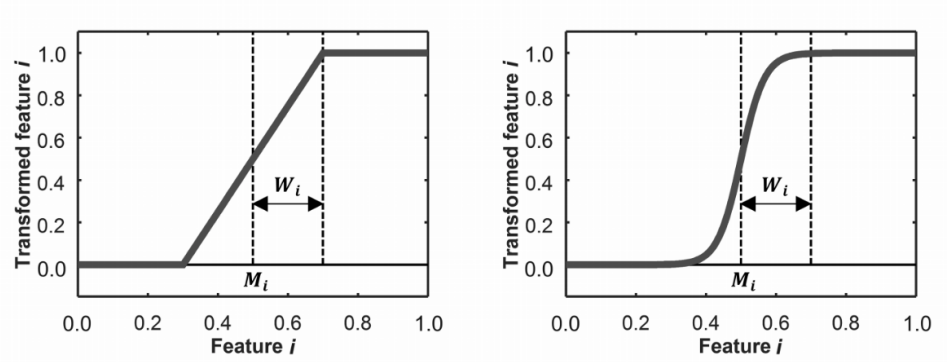

Figure 3.3: Linear (left) and S-shaped (right) transformations. M_i and W_i represent the mid-point and half-width, respectively. Figure taken from the research published by Amaya et al. [4].

3.6 Feature Transformations

There are many transformations which can be applied to the features at each state of the JSSP. As stated in Chapter 1, Section 1.2.3, Amaya et al. [4] published an extensive research in other problem domains. The main goal of the transformations is to try to separate the regions of influence of each heuristic. The ideas proposed by the author are taken, and two of the transformations used in that research are implemented for the JSS domain.

As it was mentioned in Section 2.6, this thesis is focused in the application of the ideas of these authors to the Job Shop domain. Although the authors show that kernel transformations were among the best for enhancing hyper-heuristics in the CSP domain, the scope of the research for Job Shop will be limited to the Linear (Φ_L) and S-shaped (Φ_S) transformations. The main reason behind this is that they are explicit transformations and hence their effects can be explained more easily than by using kernels. Since this is a first-hand approximation to the Job Shop domain, it was decided to focus on both of these transformations, wich are shown in Figure 3.3 and defined as follows:

$$\Phi_L(x_i, M_i, W_i) = \max\left(0, \min\left(1, \frac{x_i - M_i + W_i}{2W_i}\right)\right) \tag{3.4}$$

$$\Phi_S(x_i, M_i, W_i) = 1 - \left(\frac{1}{1 + e^{\frac{6M_i}{W_i}\left(\frac{x_i}{M_i} - 1\right)}}\right) \tag{3.5}$$

Note that they are described in terms of M_i and W_i. The first parameter is the mid-point of the transformation, and the second one the half-width. This two parameters can be furtherly defined by specifying a range on to which the transformation is to be applied. Let $[a, b]$ be such range. Then:

$$M_i = (a + b)/2 \tag{3.6}$$

$$W_i = (a - b)/2 \tag{3.7}$$

By defining Equations 3.6 and 3.7 in this way, the effects of different ranges on each feature and in the end, on the performance of the trained HH, can be explored.

The transformations are applied while the training process of the HH is being carried out. This is done by calculating the value for each feature and then apply the specific transformation to it so that the distance to the HH is measured with the set of transformed features. The same applies when a HH is tested on an instance. Each state of the solution process is transformed and then compared to the blocks of the HH.

3.7 Solution Model Overview

Taking into account what was described in the previous sections, the general process to train a single HH is shown in Figure 3.4. Each area in the block diagram is labeled with a number, which corresponds with the enumeration herein presented. Overall, to perform experiments, the following phases are carried out:

1. Define a set of instances for training the HH and another one for testing it.

2. For each instance in each of the sets, calculate an Oracle, this is, exert each heuristic over the instances and keep in the Oracle of that instance the makespan of the heuristic that achieved the lowest makespan.

3. Generate an initial HH with an equal number of blocks (rules-actions) as heuristics available (as said in Section 3.3, 6 heuristics will be used on this research). Set the parameters of the rules of each block to a randomly generated integer in the range [0,1]. Test the generated HH over each of the training set instances (the process to test a HH over an instance is illustrated in Figure 3.5). The objective function is calculated with Equation 3.2, with C_{b_i} obtained from the Oracle calculated in step 2 and determines the initial energy of the system.

4. Define the number of iterations (numit) that will be performed by the SA algorithm. The other parameters remain the same for all experiments and were defined as $T_{\text{max}} = 100$, $T_{\text{min}} = 0$, and $\alpha = -\log(T_{\text{max}}/T_{\text{min}})$. Subsequently, the SA algorithm is applied by using the generated HH from step 3 as the first candidate solution and the performance it had on the training set as the initial energy of the system. At each iteration, a new candidate solution is formed by performing any of the perturbative actions mentioned in Section 3.4. Next, the new candidate tested over each instance of the training set and the energy of the candidate solution is calculated with Equation 3.2, after this, the energy of the system is updated if the acceptance criteria are met. Next, the temperature is updated accordingly, with the exponential cooling schedule (Equation 2.2). This process is repeated until the system is cool (T_{min} is reached).

   Note that an experiment could include feature transformations. If so, for each feature of the problem's state, the transformation is applied over the original value of the feature, so that the distance to the HH's blocks is calculated between the transformed vector of features at each state and each block of the HH. The process to use a HH in a JSSP instance with and without transformations is illustrated in Figure 3.5.

5. After the SA process finishes, test the trained HH over the testing set. To measure the effectiveness of such HH, the same equation as in the last step is used (Equation 3.2).

If transformations were applied during the training phase, measure the distance of the blocks of the HH to the transformed vector of features at each state.

Finally, note that the outlined process is just for generating a single HH. Generally, throughout the experiments shown in this thesis, several replicas (usually 30) are carried out for each experiment. Hence, the process will be repeated accordingly.

3.8 Summary

The formulation of a hyper-heuristic (HH) used in this thesis consists of a set of blocks, where each block contains a set of features and an action which refers to a specific heuristic. The HH applies a different rule depending on the state of the problem, which is compared by Euclidean distance to the features part of the block. The state of a problem is defined with a set of 6 features. They are divided into two categories: features that describe the state of the schedule and features that describe the state of pending jobs. The rules in the blocks of the HH call to a heuristic from a set of 6 heuristics. The heuristics were proposed based on experiments done to include both good and bad performing heuristics. To train HHs, a simulated annealing method is proposed, the meta-heuristic follows the traditional exponential cooling schedule. Also, two feature transformations were defined that can be used during the training process of a HH. The first one is a linear transformation over the space of features, and the second one an S-shaped transformation. Both of the transformations are defined in terms of a range which will determine the area on the feature domains onto which they transform the values. An overall view of the model is described and illustrated.

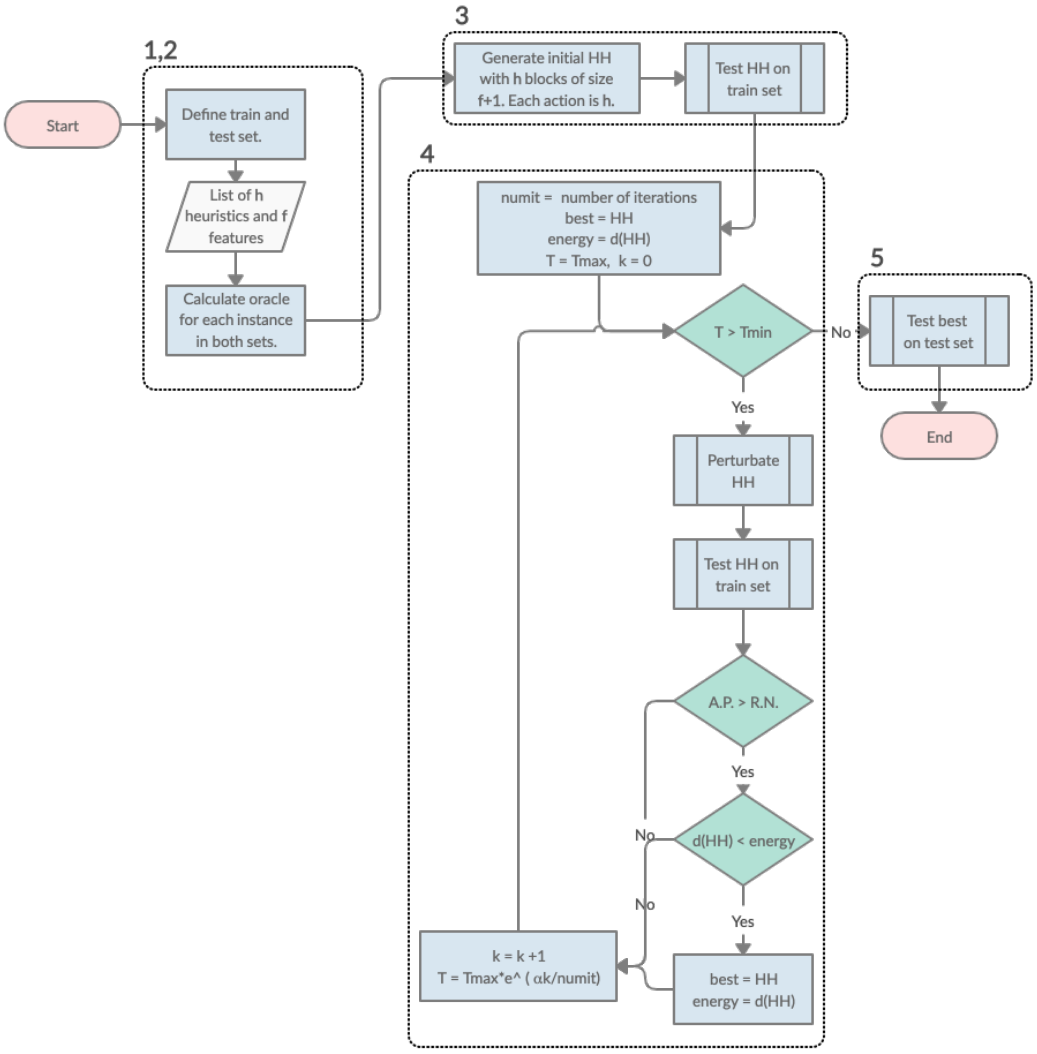

Figure 3.4: Block diagram showing the process to train a single HH. Each area has a number that corresponds to a step described in Section 3.7. Also, the diagram showing how to test a HH is shown in Figure 3.5. Note that A.P. stands for acceptance probability and R.N. for a random number. Also, the method by which a HH is perturbated is described in Section 3.4.

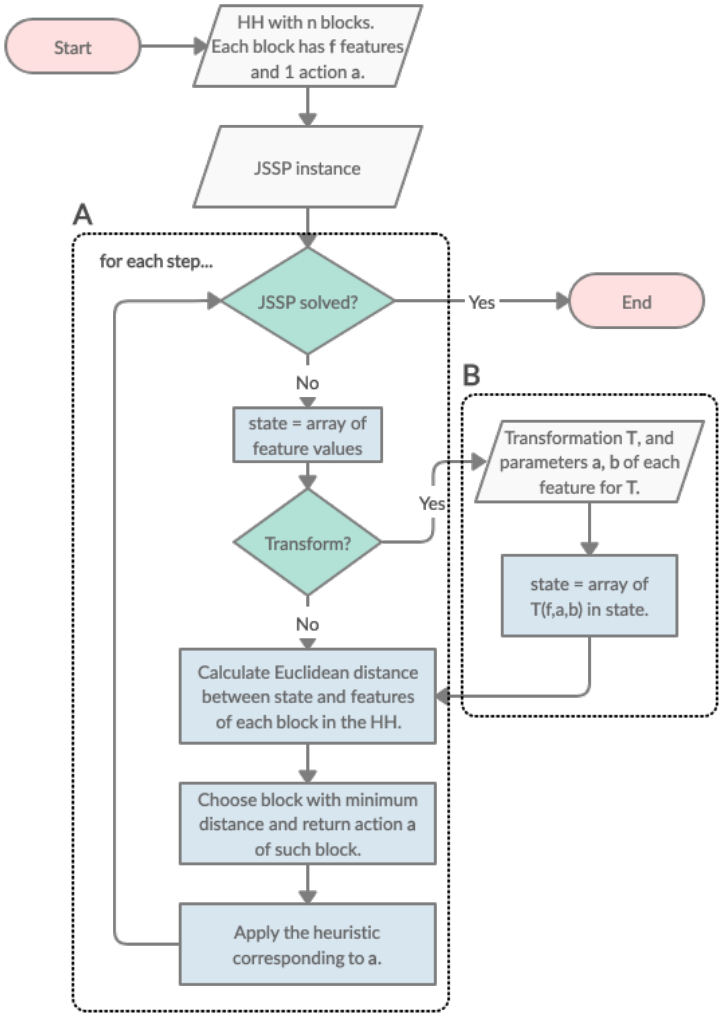

Figure 3.5: Block diagram showing the process to apply a HH to a JSSP instance. Note that area A is the general process without transformations, and area B is for the case when a transformation is applied. Hence, a HH that involves transformations goes through A and B, while one without transformations only goes through A.

Chapter 4

Methodology

This chapter describes the steps and experiments to be executed for the purposes of this research. First, a description of the framework on which the experiments are carried out is presented. Then, the definition of the different experimental phases, including the definition of the instances, replicas, and verification tests to be used at each stage.

4.1 Framework for Training and Testing HHs

A framework (HERMES) for the evaluation of constructive HHs on the CSP and Knapsack domain was provided by members of the Intelligent Systems group of Tecnológico de Monterrey. HERMES is written in Java and has been used in other works to study the effect of HHs for several optimization domains. The framework includes the necessary modules to produce HHs of the same kind (block based) as the ones proposed to use in this research (Section 3.1). Also, a genetic algorithm is provided to train the HHs, and the modules necessary to model both the Knapsack and the CSP domains are included. Members of the group have conducted different researches of selection constructive HHs with feature transformations [3, 3], but for the aforementioned domains. Because of this, the framework also provides with the modules necessary to apply them during the training process of a HH.

Since this research is focused on the JS domain, and a simulated annealing strategy will be followed, HERMES was furtherly adapted by including the necessary modules both to model the JS domain and to train HHs with a simulated annealing approach. Also, a model which includes the algorithm to adapt the feature transformations ranges (Section 7.4) was also developed and included. Finally, the framework was enhanced with some parallelization improvements, which had a positive impact in computational time for the training process of HHs. The adapted version was implemented in Python v. 3.5. by transcoding the modules in Java to this programming language and by adding the aforementioned modules. Before using the framework to execute the experiments needed for this research, several experiments were carried out to ensure that the results provided by the framework were consistent with the different constraints for the JSS domain (Section 2.2), and also to ensure that the generated HHs were behaving in the way they were expected to.

Since this is a first approach to the JSS domain with the use of HHs, it is important to note that there are many optimizations that could be done to the framework, which would

result in the possibility of doing more ample experiments, and with larger instances. Some of them are described in the last Chapter (Section 8.4.4) and are left as future work.

4.2 Experimental Stages

Several stages were defined to satisfy the objectives of the research. This section describes each of the stages and the purpose it is aiming to accomplish. Also, the experiments that will be carried out at each stage are defined, with a mention of the location of the implementation and results of each stage in the thesis.

4.2.1 Stage One: JS Domain Exploration

Three sets of experiments will be carried out to define how to properly design the confirmatory experiments of the next stages:

1. Exploration of heuristics (S1-E01) :
2. Computational time (S1-E02)
3. HH behavior (S1-E03)

This stage is implemented in chapter 05. The purpose of S1-E01 is to test the selected heuristics over different problem sizes, not only to verify that solutions are correct, but also to compare them and gain information for the interpretation of results in further stages. S1-E02 serves as a reference framework to define an approximation of the time each further experiment will take in a computer. Finally, S1-E03 will serve to explore how do the HH behaves in training and test sets, with different configurations, because there is no *a priori* evidence that a HH model will work for the JSSP.

S1-E01: Exploration of Heuristics

A set of 100 problems of sizes 5×5, 10×10, 15×15, 20×15, and 30×30 will be generated. Each heuristic will be tested on each set. A comparison will be made between the heuristics.

S1-E02: Computational Time

The objective is two-folded: first, to test whether the heuristics differ in how they behave in terms of time when applied to an instance. Second, to determine the impact training HH with SA has. For the first part, the information gained by S1-E01 will be used. For the second one, experiments of one replica and 5 iterations will be done to train the HH. The objective is to test the time each iteration consumes by varying the size of the instances and the number of instances in the set. The experiment set for this part is shown in Table 4.1.

Table 4.1: Experiments to test time complexity of the SA training process with different sizes of instances and the number of instances to be used for each set.

Size	Set 1	Set 2	Set 3	Set 4
5×5	10	20	30	40
15×15	10	20	30	40
30×30	10	20	30	40

S1-E03: HH Behavior

Because of the stochastic nature of SA, every generated HH can differ from a new one generated with the same training set. For this reason, 30 replicas of the experiment will be made. Two sets of instances of sizes 5×5 and 15×15, will be used, each with 30 generated instances. The configuration of the SA algorithm will remain constant throughout the experiments, with 100 iterations for each experiment. Instances of size 5×5 will be tested on three sets of (different) 5×5 instances, with 5, 10, and 20 instances, respectively. The same process will be followed for instances of size 15×15.

4.2.2 Stage Two: Instance Size Effect

One of the objectives of this research is to determine what is the effect of instance size and number of iteration. This section addresses such effects. For this purpose 6 experiments will be carried with two sets of instances, one of instances of size 5×5 and the other one of instances of size 15×15. The definition for each experiment is summarized in Table 4.1.

Table 4.2: Experiments for stage two with defined sizes for training, and with a test set of 5 instances of size 15×15.

E-ID	Training set	numit
S2-E01	5×5	10
S2-E02	5×5	100
S2-E03	5×5	1000
S2-E04	15×15	10
S2-E05	15×15	100
S2-E06	15×15	1000

The objective of this stage is the assessment of the following inquiries:

1. Whether there is any effect of training with small instances and testing on larger ones.
2. The extent to which the number of iterations affects the trained HH.
3. If there is evidence that applying feature transformations will aid in the training process.

Chapter 6 contains the experiments for this stage.

4.2.3 Stage Three: Effects of Transformations to Feature Space and Heuristic Paths

The objectives of this stage are:

- Confirm or reject the hypothesis posed on the previous stage, that states if the scaling of features with feature transformations could enhance the produced hyper-heuristics.
- Determine the effect of a combination of transformations in the feature has. Also, determine how do the parameters of the transformations affect the performance of the generated hyper-heuristics.
- Determine the range of influence for each transformation.

S3-E01 - S3-E09: Transforming Two Features with Different Combinations of Transformations.

Two features are selected to transform, others are left with their original values. The basis for such selection is described in chapter 7 and is an effect of some of the conclusions presented in chapter 6. The goal is to determine if the combination of transformations has any effect in the generated HH and also the extent of the impact of transforming a subset of the features in comparison to doing so for all features.

4.2.4 Stage Four: Confirmatory Experiments for Stages Two and Three

S4-E01 - S4-E04: Effect of Transformations in Instances of Size 5×5 and 15×15.

With the conclusions of the previous stages, experiments and parameter configurations for transformations are executed for instances of sizes 5×5 and 15×15, what is expected is that the key findings are consistent with the results of this experiments, otherwise possible reasons for the discrepancies are outlined.

S4-E05 - S4-E06: confirming the effects in instances of a different benchmarking set.

To reinforce the results of feature transformations, the HHs trained with instances of size 15×15 are tested over instances that come from generators which differ from the one used throughout this research (Taillard [107] ones). Such instances include 5 of size 15×15 from Lawrence [72], and 10 of size 20×15 from the set provided by Demirkol et al. [36].

4.3 Summary

The research is carried out in a modified version of a previously available framework called HERMES. The new version was rewritten in Python programming language and incorporates the simulated annealing method for training HHs and the whole framework necessary to model Job Shop scheduling problems. Also, four stages are proposed to meet with the objectives of the thesis.

The first stage is focused on exploring the JSS domain. The experiments have three main objectives, being the exploration of how the heuristics perform for several instance sizes, the computational time needed to execute those heuristics by using the framework and a first approach to the behavior of hyper-heuristics in JSS problems with the use of a random hyper-heuristic.

The second stage studies the impact that using different instance sizes have on the training process of hyper-heuristics, and also the determination of a proper number of iterations to train HHs with those sizes. In conjunction with this, it is expected that these experiments explain several aspects of the HH training process that need to be taken into consideration in order to apply feature transformations.

The third stage studies how feature transformations enhance HHs also is focused in determining the impact on the training process of HHs when doing combinations of transformations among the features, and how do the ranges of influence of the transformations have to be tuned.

The fourth stage collects all the evidence from the previous phases. Experiments are proposed to confirm the evidence in sets of representative instances for purposes of the research of the thesis.

Chapter 5

Exploring the Job Shop domain

This chapter serves a dual purpose: provide a baseline for testing the proposed solution model, and explore feasible problem configurations. Moreover, this chapter has been organized into three parts. The first one presents the performance of the heuristics considered in this work. Data are given for different instance sizes. The second part presents different aspects of the solution model in terms of computational time. This part serves to design feasible and valid experiments considering the available computational resources. The final part presents hyper-heuristics (HHs) trained under different scenarios, which will shape several aspects of upcoming chapters: number of replicas, the performance of HHs and heuristics, and the comparison of the solution model against hyper-heuristics which select an action at random. This serves to define the tests of the following experimental stages, in terms of the thesis objectives.

5.1 Behavior of Selected Heuristics

Six heuristics have been selected (Section 3.3), which represent the set of possible actions a HH can take (H). In general, several aspects of heuristics need to be defined for conclusions about their behavior to be drawn:

1. Study the makespan of each heuristic in different instances by comparing it to an approximated lower bound.

2. Ensure that the selected heuristics do not produce similar results for a range of instances that are representative of the set that will be used in further experiments.

3. Provide an example where combinations of heuristics generate better results than using the best H for the same JSSP.

The first point serves to study the performance of individual heuristics, i.e. how good each one is. The second point is included because of the model proposed to generate and train HHs. Recall from Section 1.2.2 that the basis for using a HH is that schedules with lower makespan can be generated if, for each state of the solution, the optimal heuristic for such a state is selected. If there is no variation between the decision rules used for a range of instances, then there is no point in using HHs, as it would suffice with selecting any heuristic. The third point ensures that a HH can produce better results than using the same heuristic throughout

the solution process. This does not ensure that a HH (following the proposed model) will be better, or that its generation is feasible. It only shows that, at least in theory, a HH could produce a better makespan.

5.1.1 Performance of Heuristics Against a Lower Bound (S1-E01)

To compare the performance of every $h \in H$ against the lower bound of an instance (lb), different sets of 100 instances were generated. Each one of them considered problems of a single size, among the following alternatives: 5×5, 10×10, 15×15, 20×15, and 30×30. For every set, the mean of the ratio (r_{lb}) for each instance is computed, which considers the makespan given by each $h \in H$ and by the lb (equation 3.3). Also, metric a_d is calculated, which gives the mean of the weighted average number of activities (per instance) that the heuristic increments to the schedule (compared to lb), for all instances in the set. The weight is the mean of p_{ij} (defined in Section 3.3) for the set of activities at each instance. Results are summarized in Table 5.1. Best results along all sets in terms of r_{lb} are marked in bold. Results

Table 5.1: Performance results for every $h \in H$ for 100 instances of different sizes.

Heuristic	**SPT**		**LPT**		**MRT**		**MLM**		**LLM**		**EST**	
Metric	r_{lb}	a_d	r_{lb}	a_d	r_{lb}	a_d	r_{lb}	a_d	r_{lb}	a_d	r_{lb}	a_d
5×5	1.7	11.4	1.5	10.5	**1.4**	9.3	1.6	10.6	1.6	10.9	**1.4**	9.7
10×10	1.9	24.6	1.8	23.6	**1.5**	19.9	1.8	23.3	1.8	24.3	1.6	21.4
15×15	1.9	37.8	1.9	36.8	**1.6**	30.2	1.9	36.4	1.9	37.2	1.7	32.9
20×15	1.5	37.9	1.5	36.9	**1.2**	30.2	1.5	36.5	1.5	37.3	1.4	33.0
30×30	2.1	78.8	2.1	76.1	**1.6**	60.6	2.0	74.9	2.1	78.1	1.8	67.2
Average	1.8	38.1	1.8	36.8	**1.5**	30.0	1.7	36.3	1.8	37.6	1.6	32.8

indicate that in general, heuristics are far from the estimated lower bound. This is not a matter of concern since lower bounds are approximate. Should heuristics reach the lower bound, it would mean that the heuristic yields an optimum makespan, which is not something to expect from dispatching rules.

MRT is the heuristic that performs best in terms of makespan for all the tested sets, closely followed by EST. In fact, they were tied in instances of size 5×5. Moreover, as instances grow in size, the gap in performance tends to become larger. Also, there is a tendency for heuristics to diverge more from the lower bound. This can be seen more clearly when metric a_d is analyzed because it represents how many activities of average processing time are incremented from the makespan reported by the lower bound. Nevertheless, and as aforementioned, this does not say too much about whether heuristics are good or bad since they are not being compared to a true optimum.

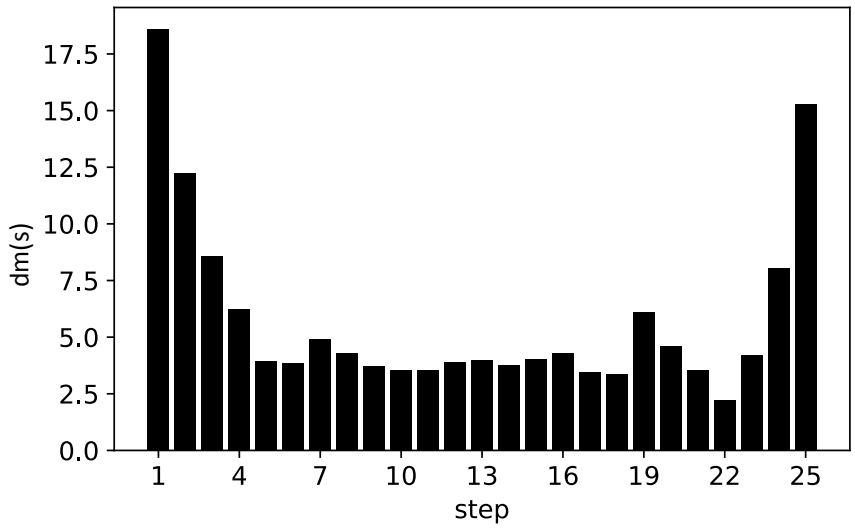

Figure 5.1: Average of $dm(s)$ (given in Equation 5.1) at each step of the construction of a schedule for instances of size 5×5.

5.1.2 Diversity Within Heuristics

This thesis proposes a test to measure heuristic diversity: For a given instance, determine the number a_s of heuristics that have the same action (i.e. choose the same activity) at each step s of the solution. Thus, the ratio of such heuristics can be calculated for every instance, and the mean ratio can be used as a diversity metric (Equation (5.1), where H is the set of all heuristics). Striving to test this idea, experiments are carried out considering all heuristics, operating on sets of 100 instances whose size can be either 5×5 (Figure 5.1) or 15×15 (Figure 5.2).

$$dm(s) = \frac{\sum_H a_s}{|H|} \tag{5.1}$$

Data show that heuristics tend to follow different paths on both sets. However, it is interesting to see that heuristics behave alike at the beginning and at the end of the solution. The effect at the end is likely to happen because it is expected that the set of pending jobs is reduced towards the end of the solution of an instance. Also, the effect lasts longer in the set of smaller instances. This indicates that there may be characteristics inherent to the instance size in relation to the heuristics' behavior which should be studied.

5.1.3 Exploring Combinations of Heuristics

Recall from Section 3.1 that a hyper-heuristic (HH) selects a specific heuristic based on the feature values of the current problem state. For hyper-heuristics to work on the Job Shop domain, it is necessary to verify the existence of cases where the combination of heuristics yields better results than standalone heuristics. To do so, experiments were carried out in the same sets of instances from the previous section. At each step of the solution, the HH chooses a random heuristic.

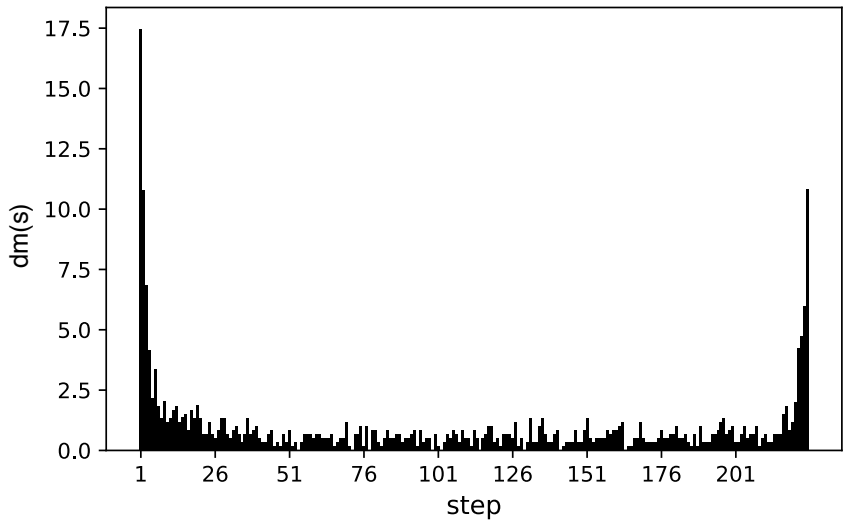

Figure 5.2: Average of $dm(s)$ (given in Equation 5.1) at each step of the construction of a schedule for instances of size 15×15.

Although this approach does not use a trained hyper-heuristic, it offers a quick way to determine if combining heuristics is feasible. Then, for each result, an analysis is made on the number of times that using the random heuristic (RHH) yielded a lower makespan than those given by the set of heuristics (Tables 5.2 and 5.3). Also, to measure the impact of the RHH (either positive or negative), equation 3.2 is used, where RHH refers to the hyper-heuristic (HH). Although both sets delivered a rather small average makespan difference (d_w), data serve as a proof-of-concept that by combining different heuristics, it is possible to find a better schedule than by using the heuristics directly.

Results indicate that for both sets, there are cases where the RHH wins against the single use of heuristics. Nevertheless, there are two important aspects of this conclusion. First, only 16-17% of the RHHs outperform the oracle. Second, that the difference on the solutions provided by the winning RHHs and the best heuristics is small (-0.8 % for 5×5 instances and -0.5 % for 15×15 ones). Both of these aspects do not necessarily imply that a hyper-heuristic approach is not promising, because even though the difference against the oracle is small, hyper-heuristics could perform as well as the oracle, with the additional capability of being able to generalize over a broader set of instances. Also, this is a baseline of random hyper-heuristics, this means that they are not taking into account a specific state of the problem. Because of this, the model proposed in Chapter 3 to read states of the problem and to train HHs accordingly seems necessary in order to produce better results.

5.2 Analysis of Computational Time (S1-E02)

The first test is to analyze the time it takes for each heuristic to solve a problem. Note that this time analysis is relevant because of the framework being used (Section 4.1) and the computational limits it imposes to the experiments. The aim of this analysis is not to study the

Table 5.2: Comparison of the random hyper-heuristic (RHH) against the best makespan for instances of sizes 5×5 and 15×15. w: number of times RHH won, d_w: mean difference of instances where RHH won, t: number of times RHH was tied, l: number of times RHH lost, d_l: mean difference of instances where RHH lost, $\max_d$: maximum difference throughout the set, $\min_d$: minimum difference throughout the set.

Set	**w**	d_w	**t**	**l**	d_l	**max**$_d$	**min**$_d$
5×5	16	-0.0086	7	77	0.1088	189	-52
15×15	17	-0.005	0	83	0.069	284	-129

Table 5.3: Number of times that each heuristic was selected during the solution process. Data given corresponds to instances where the random hyper-heuristic (RHH) outperformed the best standalone heuristic.

Size	**SPT**	**LPT**	**MRT**	**MLM**	**LLM**	**EST**
5×5	62	74	71	59	58	81
15×15	651	609	641	633	594	697

computational aspects of the heuristics by themselves, but to be able to determine how much time is needed to construct complete solutions by using a single heuristic.

Figure 5.3 presents the results for the datasets described in Section 5.1.1. All heuristics behave similarly in small problems. Notable differences in times across the heuristics only arise when testing in instances of sizes 30×30, and it is expected that this behavior continues as instances grow.

Data corresponding to the experiments shown in Table 4.1 is summarized in Table 5.4. Since the training of a hyper-heuristic is a stochastic process, several repetitions must be made before analyzing the data. The data shown is the average time of one repetition calculated with the results of 30 repetitions. Results show that computational time grows exponentially as instance size and number increase.

Table 5.4: Average time (in seconds) required for training with SA throughout 5 iterations, for a different number of instances of different sizes. The marked cell corresponds to the configuration of training experiments that will be taken into account as an upper limit throughout the thesis.

	Instance size				
# instances	5×5	10×10	15×15	20×15	30×30
10	0.7303	9.035	50.659	100.54	1112
20	1.855	15.777	122.20	179.72	2156
30	2.081	23.675	<u>144.93</u>	336.79	3876
40	2.765	44.979	193.97	493.28	7146

* In 4-core Intel Core i7 processor @ 3.1 GHz.

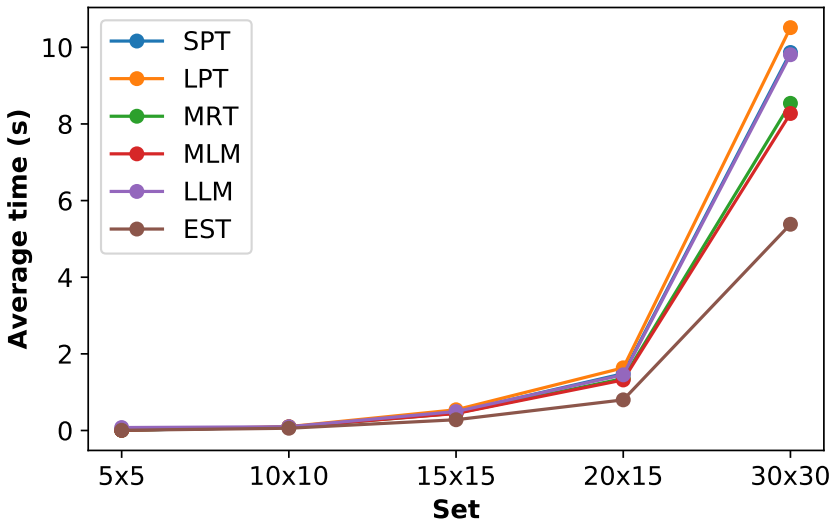

Figure 5.3: Average time (in seconds) that each heuristic takes to construct a solution for sets of 100 instances of different sizes.

All tests presented in this section were executed on an Intel Core i7 computer @ 3.1GHz (4-cores). Also, the only operation that is explicitly parallel in the framework is the calculation of energy during the SA training process. Recall that this implies calculating d for each instance of the training set. In doing so, the number of problems that the HH simultaneously solves is equal to the number of available cores.

With the computational resources available it was established that for the purposes of this research, and in views of carrying out the experiments needed to fulfill the objectives of the thesis, 30 instances is a manageable set, with instances no bigger than 15×15.

5.3 Behavior of Hyper-heuristics (S1-E03)

Results from the experiments described in Section 4.2.1 are illustrated in Figures 5.4 and 5.5. As it was mentioned in the previous section, the stochastic nature of SA demands that the training of each hyper-heuristic (HH) is repeated several times, as performance will differ from one repetition to the other. For this reason, 30 replicas of every experiment were executed. The training set contains two sets of 30 generated instances, one of size 5×5 and one of size 15×15. The SA algorithm was used with 100 iterations. After training a HH, it is tested on a different set of instances with the same size. Each test set comprises 5 (T5), 10 (T10) and 20 (T20) instances. The same process is carried out for 15×15, and testing accordingly, in sets with instances of size 15×15.

Figures 5.4 and 5.5 show the resulting data for HHs trained on instances of size 5×5 and on instances of size 15×15, respectively. In general, 100 iterations seem to generate HHs with more variance. This is more clearly seen in Figure 5.5, where T10 and T20 exhibit a variation of around 0.15. Nonetheless, it is because of that variance that HHs can outperform the oracle in some cases, although small in number, and with little effect. For instances of size

5×5 there seems to be a tendency to diminish the generalization capacity as more instances are added to the test set. At this point, it is evident that further experiments are needed to assess the effect that scaling has on the JSSP. Nevertheless, these experiments confirm that HHs can be successfully applied to the Job Shop problem domain.

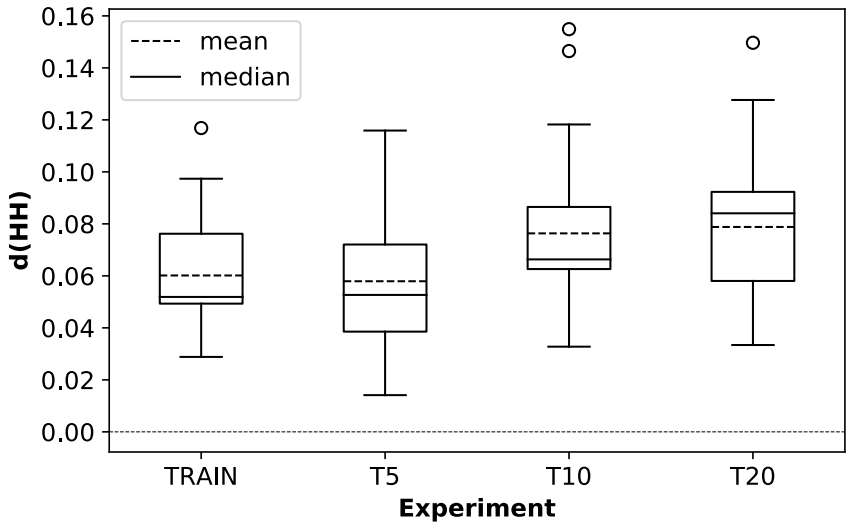

Figure 5.4: Performance of hyper-heuristics (30 runs) trained with 30 instances of size 5×5 (TRAIN), and tested on a set of 5 (T5), 10 (T10) and 20 (T20) instances of same size.

5.4 Summary

The main goals of this Chapter were to present the expected behavior of the heuristics chosen in this research; to analyze the computational time resources that will be needed for further experiments; and to establish a baseline of random hyper-heuristics for the JSS domain which serve as a starting point to verify that the solution model proposed could indeed produce HHs which produce reasonable solutions when compared to the single heuristics.

On the behavior of heuristics experiments with sets of 100 instances of 5 different sizes were carried out. The performance of the heuristics was compared towards a theoretical lower bound, calculated for each instance. Results indicate that the MRT and EST heuristics are overall better than the other heuristics. Also, that the heuristics' performance is inversely proportional to the size of an instance, with the exception of instances of size 20×15. Besides this, a diversity measure was used to study the similarity between the heuristics. The data indicate that in an average of 100 instances, the heuristics tend to choose similar activities in the first and last steps of the solution process.

Also, a random hyper-heuristic was used to establish if the idea of using different heuristics at each step of the solution process is able to produce better results than using the same heuristic throughout the construction of the schedule. Results indicate that for 15% of the cases on average, the random hyper-heuristic outperforms the oracle.

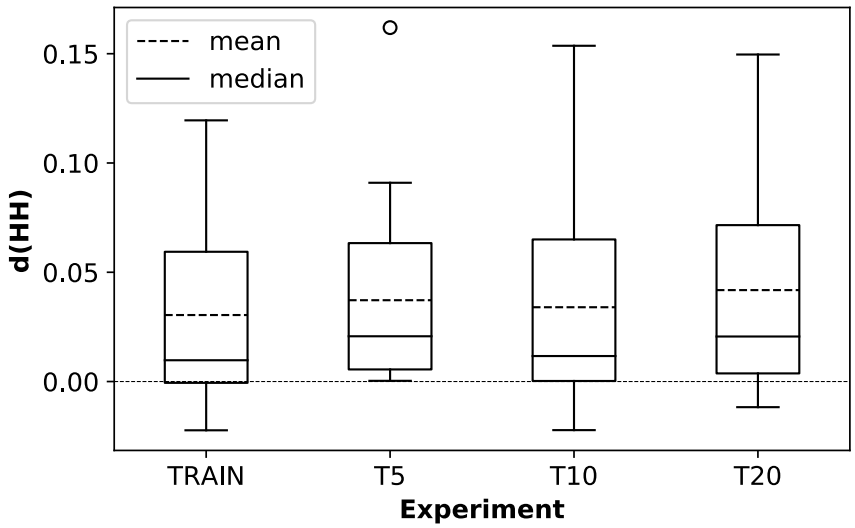

Figure 5.5: Performance of hyper-heuristics (30 runs) trained with 30 instances of size 15×15 (TRAIN), and tested on a set of 5 (T5), 10 (T10) and 20 (T20) instances of same size.

Finally, the simulated annealing methodology is tested. An analysis of computational time needed to execute one iteration of the process served to establish methodological aspects of further experiments. Sets no bigger than 30 instances and of sizes no bigger than 15×15 were defined to execute the further experiments. Also, a first approach of the solution model was carried out with 100 iterations and instances of size 5×5 and 15×15. The results indicate that for some cases of the instances of 15×15, the HHs are able to outperform the oracle, but also that HHs have more variance. These results endorse the idea that using the solution model proposed could yield promising results in the JSS domain.

Chapter 6

Assessing the Performance of Hyper-heuristics for the JSSP

As it was mentioned in the previous Chapter, as the problem instance grows in size, the amount of resources required to compute a solution also increases. Thus, the first intuition about how to train a hyper-heuristic (HH) may be to use instances with the same size for training and testing. The idea with this approach is to preserve the nature of both problems as similar as possible. This Chapter shows how the instance size impacts training, and whether there is a difference in performance when training with instances of one size or the other. Moreover, an explanation for such an effect is given. This serves as a basis for applying feature transformations on the next chapter.

6.1 Effect of Problem Complexity on Heuristics and Features

Recall (from section 2.2) that the constraints inherent to an instance deeply affect the solution process and the dependence of each subproblem. Training the HH with Simulated Annealing (SA) implies that, in theory, the blocks should tend to replicate or improve the best heuristic for a specific state of the problem. The way in which the HH 'views' such a state is through the set of features. Several assumptions are required for such an argument to be valid, namely:

1. Features can describe different states of the problem.
2. Simulated Annealing can generate HHs with parameters sufficiently spread throughout the domain of each feature.
3. Heuristics are diverse enough throughout the solution of a JSSP so that blocks within the HH can represent such differences.

Suppose that feature values throughout training are very similar no matter which heuristic is used. This would negatively impact the ability of SA to find a good HH since the aforementioned assumptions do not hold. In this particular case, the problem is that all heuristics

point towards the same decisions, so the same path will be followed no matter which rules are defined.

Let us measure how many possible solution paths remain available when a decision is taken. Let u_i^h be the number of distinct jobs available for the set of heuristics (H) when the heuristic h assigns an activity at step i. Also, let p_i^h be the number of pending jobs at step i when solving with heuristic h. Then, the average ratio of u_i^h and p_i^h for every h in H shows the diversity of jobs across the solution of an instance. The smaller the measure, the more diverse paths are for that step. Conversely, the larger the measure, the fewer solution paths are available. Figure 6.1 shows the result of doing said analysis on a 15×15 instance. It is clear that diversity tends to increase as the solution progresses.

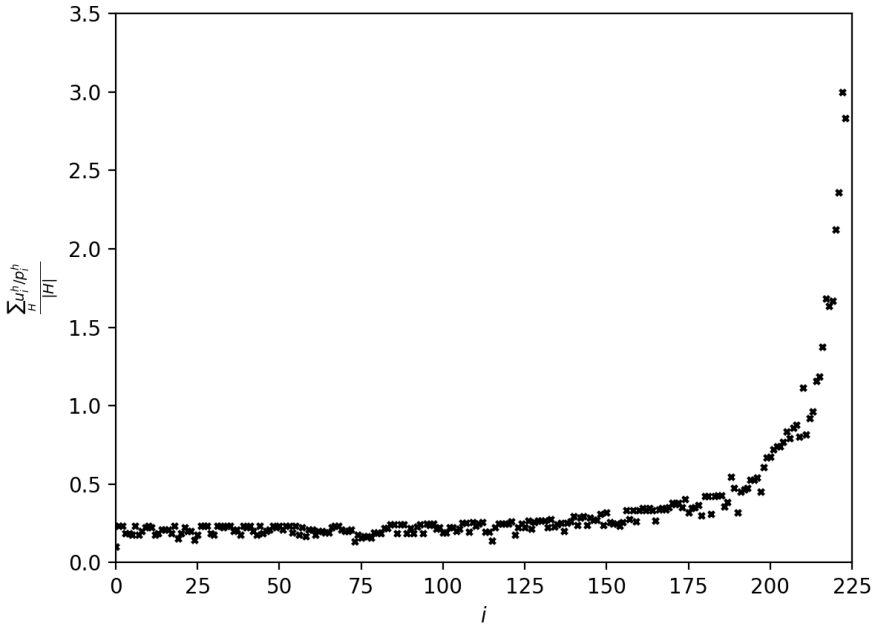

Figure 6.1: Average of u_i^h/p_i^h for all $h \in H$ of a 15×15 instance.

It is important to note that u_i^h/p_i^h is not to be confused with the ratio of unique and pending jobs. Should this be the case, the range of the measure would be [0,1]. What is being measured here is the impact on the diversity of jobs to be scheduled for all heuristics, whenever each heuristic tries to solve the problem. This implies that the decision of scheduling a job greatly restricts other possibilities until near the end of the process. Hence, it limits the possibility of a heuristic to achieve a better result. In some way, the paths are being constrained to previously selected heuristics. This may hinder the training process. During the first stages of the solution process, possible solution paths are small relative to the final phase of the solution. In other words, no matter how good the training process is, the extent to which the HH will produce a better solution is greatly limited by the heuristic it chooses to apply at the first steps of the solution. This directly affects the validity of the assumption (2).

Since the hyper-heuristic chooses a heuristic by measuring the similarity of the current problem state and its blocks, the ability of the features to represent small changes between

states produced by one heuristic or another is important. This condition can be tested by measuring the diversity of feature values for each heuristic throughout the solution process. Figure 6.2 shows such data. It can be seen that for most feature values (i.e. all but a), at the beginning of the solution there is no significant difference across heuristics. This, in turn, will affect the training of a HH as aforementioned: the HH will have an almost predefined performance which will depend on the probability of generating sufficiently dissimilar parameters for each feature. Hence, should feature values not be dispersed enough for each heuristic, especially at the first steps of the solution process, the performance of the HH will not be good.

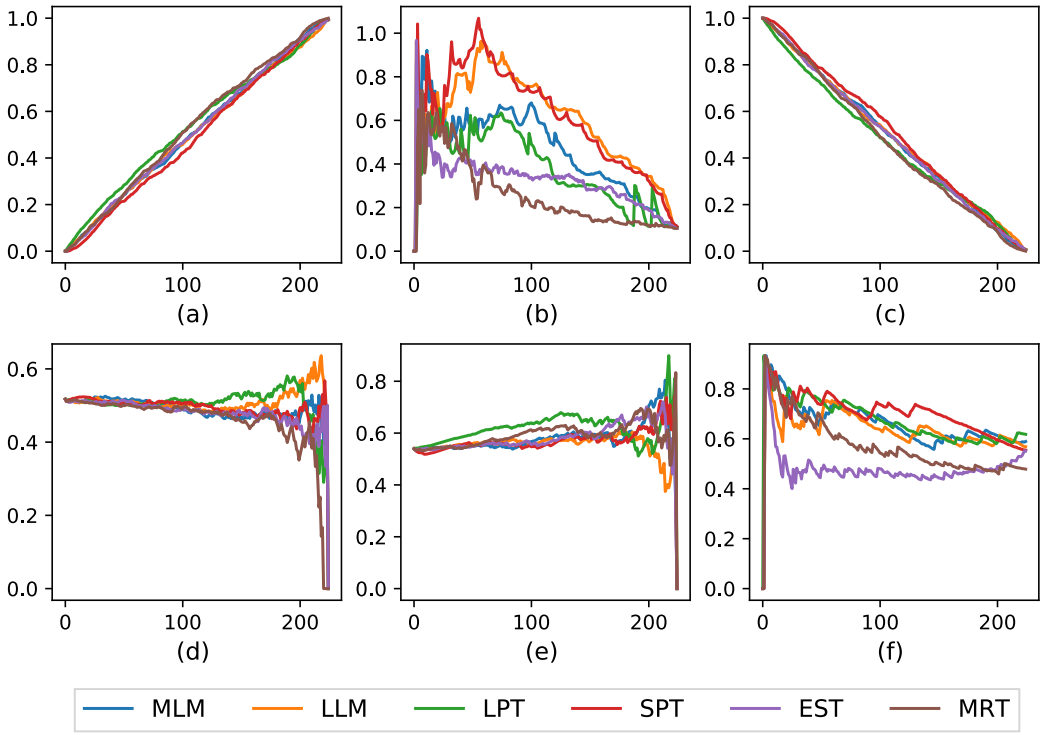

Figure 6.2: Feature values for an instance (size 15×15): (a) APT, (b) DPT, (c) NAPT, (d) NJT, (e) DNPT, and (f) SLACK. In each subplot, every heuristic is plotted with a different marker. The horizontal axis corresponds to each step of the solution process.

This work proposes that such dispersion can be increased in three ways:

1. By providing instances with more dispersion on the feature values: The size of the instance is directly correlated to the number of steps a heuristic (or HH) has to take to construct the solution. In fact, this amounts to $m \times n$ steps. Values for each feature are calculated with ratios that depend on the problem size (refer to section 3.2). Nonetheless, to assess whether dispersion between values is directly or inversely proportional to instance size, further experiments have to be carried out.

2. By allowing SA to iterate more: Since the process is stochastic, the probability of generating better parameters increases.

3. By performing feature transformations: A mapping function can be designed so that small differences between feature values are enhanced.

For the purposes of this chapter, methods (1) and (2) will be studied. The study of the method (3) is left for chapter 7.

6.2 Effect of Instance Size (S2-E01 - S2-E06)

The previous section laid out the question about how problem size impacts training performance. To assess this effect, the following experiment is presented:

1. Train a hyper-heuristic (HH) with 30 instances of size 15×15 with 100 iterations and test it on a (different) set of 5 instances of the same size.

2. Train another HH with 30 instances of size 5×5 with 100 iterations and test it on the testing set used in the previous step.

3. Execute 30 replicas of steps 1 and 2.

4. Evaluate the differences between the results for the testing set.

Figure 6.3 illustrates the resulting data. The HHs generated with smaller instances performed slightly better than the ones generated with larger instances. This experiment shows a confirmation of the first proposed way to increase dispersion (Section 6.1). During this research, it has been noted that feature values in smaller instances tend to me more disperse than feature values in larger instances. To illustrate this statement, a comparison can be made between the feature values of instances with different sizes. Since features change throughout the construction of a solution, it is convenient to choose features that do not rely on the state of a solution but on the state of the problem. In this way, a comparison can be made on feature values for different instances before they are solved. From the set of features used in this research, features NJT and DNPT have the said characteristic. Both of them represent the state of pending jobs, hence their values are different for each instance, no matter if the solution has been constructed yet. Figure 6.4 shows the NJT and DNPT values of instances of three different sizes. It is clear that the values of features are more dispersed in smaller instances than in larger ones. Considering this, it can be concluded that because smaller instances have more dispersed feature values, the HH training process is able to produce blocks that overall affect a larger area of the domain of each feature, therefore, it can be said that the blocks are more representative of a problem state than those produced with larger instances. Nevertheless, this result is limited by the fact that 100 iterations were used to train the HHs. As said in Section 6.1, the SA process is stochastic, so the number of iterations used for larger instances may not be sufficient for the HH training process to generate blocks that are sufficient in terms of representing the domain of features. The next section presents the experiments and discussion of these implications. Also, a study involving both the number of iterations and instance sizes needs to be exerted to study how is the relationship between both effects. This is presented furtherly in Section 6.4.

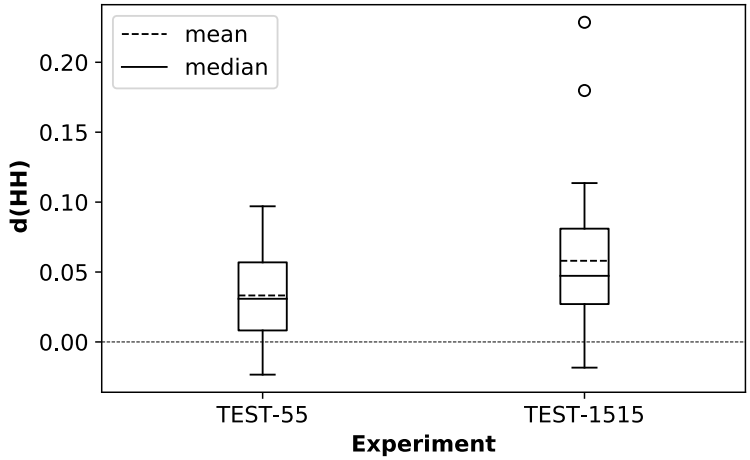

Figure 6.3: Performance of hyper-heuristics (30 runs) over a test set with 5 instances of size 15×15. Hyper-heuristics were trained on 30 instances of size 5×5 (Left) and of size 15×15 (Right) with 100 iterations. $d(HH)$ is given in Equation 3.2.

6.3 Effect of the Number of Iterations when Training HHs in Relationship to the Instance Size

Section 6.1 proposed that besides the instance size (as described in the previous section) another way of improving HH training was by performing more iterations. The argument behind that idea is that by allowing the SA process to traverse with more steps the space of combinations of blocks for HHs, there is a higher probability that the generated blocks are more representative of the domain of each feature, and also could produce HHs that better represent the states of a solution. To test this idea, an experiment was carried out with 30 training instances and 5 testing instances. Both sets comprised instances of size 15×15. The experiment was done at three levels: 10, 100 and 1,000 iterations. Figure 6.5 shows that by increasing the number of iterations, the algorithm achieves better results for both, training and testing phases. When performing a small number of iterations, the stability of the generated HH is small. Perhaps the most notable difference happens between 100 and 1,000 iterations, where the generated HH even improve the results that the best heuristic achieves. A comment about the number of iterations and stability is important. In this experiment, for the 5% best HH trained with 1,000 iterations, the energy becomes stable -on average- with 720 iterations. And for the 5% of the worse on an average of 150 iterations (refer to Appendix B for an illustration of this effect). Regarding what was said on the previous section about the number of iterations, it seems that 100 iterations are not sufficient for training HHs of size 15×15, but 1,000 seems reasonable, considering the computational resources available, and the time it takes to compute each iteration (Table 6.1). Although improbable, a case could arise on which a HH has better performance with more than 1,000 iterations, but making further studies with the computational resources the author has at hand is intractable. Section 6.2 showed that HHs trained with 5×5 instances had even slightly better performance as the ones trained with instances of size 15×15. Nevertheless, in order to conclude to the reasoning that has been

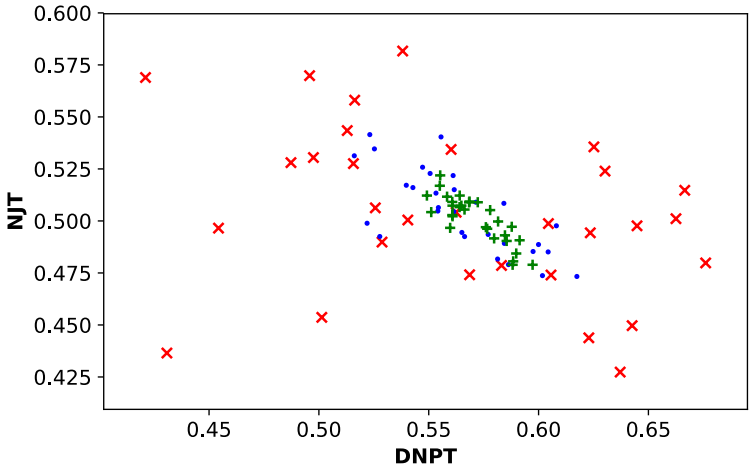

Figure 6.4: Feature values (DNPT and NJT) for sets with 30 instances of different sizes. Red 'x' markers: 5×5 instances; Blue '.' markers: 15×15 instances; Green '+' markers: 30×30 instances.

followed throughout this Chapter regarding the ways by which dispersion of feature values for the different heuristics can be augmented, there is still the open question whether if HHs trained with instances of the same size as the testing set and a sufficient number of iterations (for 15×15 this number seems to be 1,000) exhibit a performance similar as that of HHs trained with smaller instances and that same number of iterations when being tested on the set of larger instances. The next section shows experiments involving both the effect of the number of iterations and the sizes of the instances have on the performance of trained HHs.

6.4 Combined Effect of Instance Size and SA Iterations

Section 6.2 hinted at the possibility that, for some cases, training with instances of a smaller size makes no significant difference on hyper-heuristic performance ($d(HH)$) when being tested on sets of larger instances and in comparison with performance of hyper-heuristics trained with instances of the same size as the test set. Nevertheless, Section 6.3 showed that for instances of size 15×15, the number of iterations used in the experiments carried out in Section 6.2 was not sufficient for the HHs trained with instances of that size, and that to compare those against the HHs trained with instances of smaller sizes, this fact had to be taken into consideration. Moreover, training on larger instances requires more computational resources. Hence, it is important to study whether training with small instances for more iterations leads to acceptable performance in terms of the performance achieved by HHs trained with instances of larger size when both sets of HHs are tested on a set of 15×15 size. Figure 6.6 shows data at three levels of iterations (10, 100, 1,000) and for two instance sizes (5×5 and 15×15). Every HH was tested on a set with instances of size 15×15. Clearly, the best results were achieved by training for 1000 iterations and instances of size 15×15. But an interesting case arises: Data for 55-TR-1000 and 1515-TR-100 looks quite similar, and it seems like 55-TR-1000 is overall better than the other one when being tested on the set

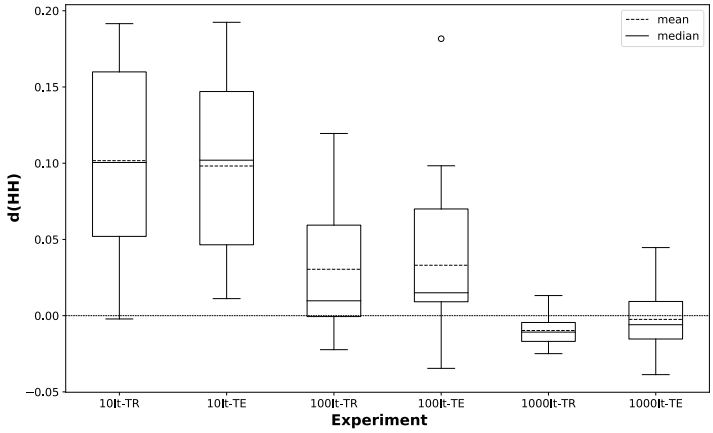

Figure 6.5: Performance of hyper-heuristics (30 runs) with a different number of iterations: 10, 100, 1000. Data is shown for training (TR) and testing (TE). In all cases, 30 instances of size 15×15 were used for training. $d(HH)$ is given in Equation 3.2.

of larger instances than 1515-TR-100 when being tested on that same set. This indicates that both the instance size and the number of iterations are helping to enhance the performance of the HHs, to the level that for 100 iterations, they even outperform the HHs generated with instances of the same size as the test set. Also, although the performance of 1515-1000-TR HHs is better than the one produced by the 55-1000-TR, what is gained in the performance of the 1515-1000-TR HHs, is lost in terms of computational time. Table 6.1 shows that the time required to train HHs with 1,000 iterations and instances of size 5×5 (6.5min) is way below the one for training with 15×15 instances (2.5h). This is interesting because this effect could be exploited when dealing with limited computational resources. The experiments carried out throughout this Chapter confirm the idea that augmenting dispersion between future values is beneficial to the training process of an HH. Also, the first two ways presented in Section 6.1 for aiding the HH training process were explored, and the results indicate that what was introduced as means of theoretically augmenting dispersion seems to be coherent with what was found in the experiments. In fact, it was also found that HHs trained with smaller instances can have a similar performance on sets of larger instances as HHs trained with instances of the same size of the testing set. The next Chapter deals with feature transformations, and the ideas of instance size presented in this Chapter are put to use to explore the extent to which feature transformations are able to enhance HHs. The results of this Chapter reaffirm the idea of augmenting features' values dispersion and indicate that using feature transformations seems like a good idea to enhance HHs; also, the benefit in terms of computational time and comparable performance achieved with HHs trained with 5×5 instances is used to be able to make more experiments regarding feature transformations than the amount that could be done if working solely in instances of size 15×15.

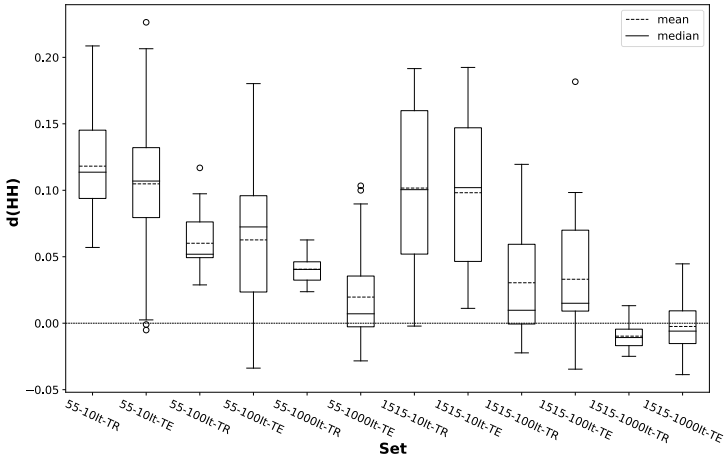

Figure 6.6: Performance of hyper-heuristics (30 runs) with a different number of iterations: 10, 100, 1000. Data is shown for training (TR) and testing (TE). Each hyper-heuristic was trained with 30 instances of sizes 5×5 and 15×15. $d(HH)$ is given in Equation 3.2. Statistical tests are reported in Appendix A, Table A.1.

Table 6.1: Time required to train a hyper-heuristic with 30 instances of sizes 5×5 and 15×15, with a different number of iterations.

Size	**Iterations**	**Time (per run)**
5×5	10	6s
5×5	100	45s
5×5	1,000	6.5min
15×15	10	3min
15×15	100	15min
15×15	1,000	2.5h

6.5 Summary

The Chapter starts by presenting a rationale that clears how the complexity inherent to JSS problems impact the way by which similar solution paths are taken when using a specific heuristic to solve the problem, and paves the way into the main reasons why the paths of feature values that each heuristic produces need to be sufficiently separated between them to allow the HHs training process to distinguish between the states of a problem and hence, produce blocks which are dealing with specific parts of the solution of a JSSP. In general terms, this research focuses on finding ways to produce a higher level of dispersion between the path of features for the heuristics. In this manner, three ways are proposed to produce the aforementioned dispersion.

The first way has to do with instance size. Experiments show that smaller instances have feature values which are more separated between them than feature values of larger instances.

HHs were trained with this consideration, and results indicated that training with smaller instances produces better results for a small number of iterations than HHs trained with instances of larger size. Furthermore, for a larger number of iterations, the results achieved by smaller instances were comparable to those produced by the HHs trained with larger instances. Also, the computational time required to train HHs with small instances is considerably less than that of HHs trained with larger ones.

Overall, the experiments carried out in this Chapter not only discover the idea of using smaller instances to study the HHs for the JSS domain, which allows a larger set of experiments taking into account the computational resources available but also confirm that by augmenting dispersion of feature values produce better results in the trained HHs.

Chapter 7

Feature Transformations

This chapter studies the effects of transforming feature values through two different functions. The main goal is to induce a larger separation of feature values for each heuristic. The chapter begins by defining the transformations that will be used. Afterward, it shows an exploratory study to identify how transformations affect each feature. With this information, it becomes natural to assume that tuning transformation parameters may improve hyper-heuristic (HH) training. This chapter then proposes and tests a methodology to adjust such parameters. Finally, towards the end of the chapter, experiments about the application of tuned feature transformations are shown.

7.1 Rationale Behind the Use of Feature Transformations

The previous chapter stated that one way for improving dispersion of feature values is to transform them. The arguments behind this reasoning were briefly introduced in Chapter 1 (Section 1.2.3), and furtherly developed in Chapter 6. The main reasons for using feature transformations have to do with the scaling effect on feature values that was illustrated in the last Chapter by using smaller instances. The expected effects that transformations have on feature values can be summarized as follows:

1. Reducing the likeliness between solution paths covered by each heuristic, hence allowing the HH to better distinguish the effect of each heuristic.

2. Enhancing the feature space, so that similar states are better represented and no rules with different actions occur for a similar state.

The previous chapter showed an illustrative example of the purpose of reason (1). Figure 6.1 showed that whenever a job is scheduled by a heuristic, the amount of feasible candidate jobs for scheduling next is reduced significantly. This affects the HH training process since it becomes more difficult to decide whether one heuristic or the other should be used. Although this argument provides the intuition that likeliness of heuristics is inevitable (for example, in the final steps of the solution process, when a small number of jobs are pending to be scheduled), it is necessary to determine if by modifying the values of the features the HH training process can, in fact, reduce this likeliness effect. Heuristics produce a specific path of solution. If for the set of features chosen, the heuristics produce similar values for the paths

(although solutions paths are different), then the HH training process will have a hard time to distinguish between the use of one or another heuristic, because feature values produced by the heuristics are similar.

Both effects enunciated above were studied by Amaya et al. [3] for the knapsack domain. The author refers to them by employing the terms likeliness (effect 1) and stagnation (effect 2) and uses the concept of a zone of influence to explain the extent to which the features are correlated to the actions. The zone of influence of a heuristic is the region of the feature space where the HH applies such a heuristic. In other words, the domain of the combined space of features for each heuristic. Based on this idea, it is convenient to study how a given transformation affects the feature space for the JSS domain.

7.2 Effect of Transformations in Feature Values

Recall, from the definition of linear (LT) and s-shaped (ST) transformations given in Chapter 3 (Section 3.6), that transformations can be tailored by adjusting their parameters. This allows shifting the transformation so that it fits a given range in the feature space. The feasible interval for each feature is in the range [0,1]. This section focuses on studying if transformations produce any noticeable effect on the features, regarding the issues presented in the previous section. For doing so, it is important to consider that feature values are constantly changing when the instance is being solved (as shown in Figure 6.2), but, there is a difference between features that describe the state of jobs yet to be scheduled, and those that describe the state of already scheduled jobs. The first one relates to the part of the problem to be solved, while the second one represents the solution. Hence, to determine if transformations help overcome the aforementioned negative effects, features of the first kind are selected. By doing so, an assessment of the effects of transformations can be made without solving the problem. Of the features proposed in Chapter 3 (Section 3.2), the ones that meet this condition are NJT and DNPT. The former describes the average pending processing times per job, while the latter represents the dispersion of processing times of activities. In both cases, for jobs/activities that are waiting to be scheduled.

With these features, the ST was applied with a range of influence of [0,1] to a set of instances. For this range, LT has no effect on the features (since it is a direct mapping of the whole range), so it was disregarded. Figure 7.1 shows that when applying the transformation, the effect of dispersing the feature values for each of the instances is achieved.

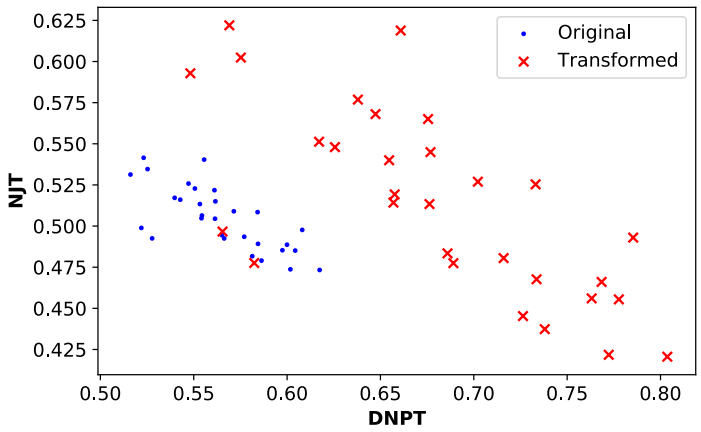

Figure 7.1: Original (marker '.') and ST transformed (markers 'x') values of DNPT and NJT features for 30 instances of size 15×15.

Nonetheless, it is also important to separate the paths of feature values that each heuristic generates. Since feature values change for each heuristic's solution, they can be compared against their transformed values to illustrate the actions transformations exert on the heuristic paths. Figure 7.2 shows the comparison between original and ST transformed values for each feature. Also, each heuristic path is shown per feature. Although the separation between paths is not very clear in all cases, the spread in cases (b),(c), and (d) is notorious. Similarly, Figure 7.3 shows the effect achieved with the LT. In this case, as it was expected, results show no significant difference between paths. Nevertheless, this does not necessarily mean that the transformation is useless. Consider, for example, Figure 7.4 where original and transformed values of APT are shown. In this case, LT transformation with a range of [0,0.1] was applied, and it improved the spread. Hence, additional tuning should be done for each transformation to detect the best parameters for each feature.

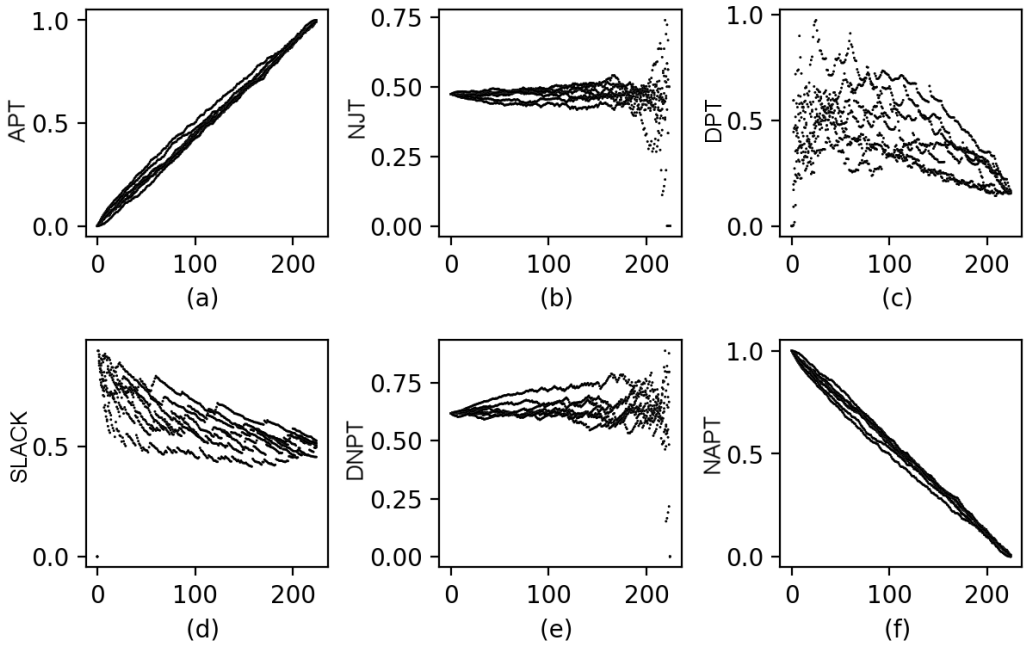

Figure 7.3: Original (gray) and LT transformed (black) feature values for all heuristics when solving an instance of size 15×15. Horizontal axis of each sub-plot is the step. Features: (a) APT, (b) NJT, (c) DPT, (d) SLACK, (e) DNPT, and (d) NAPT.

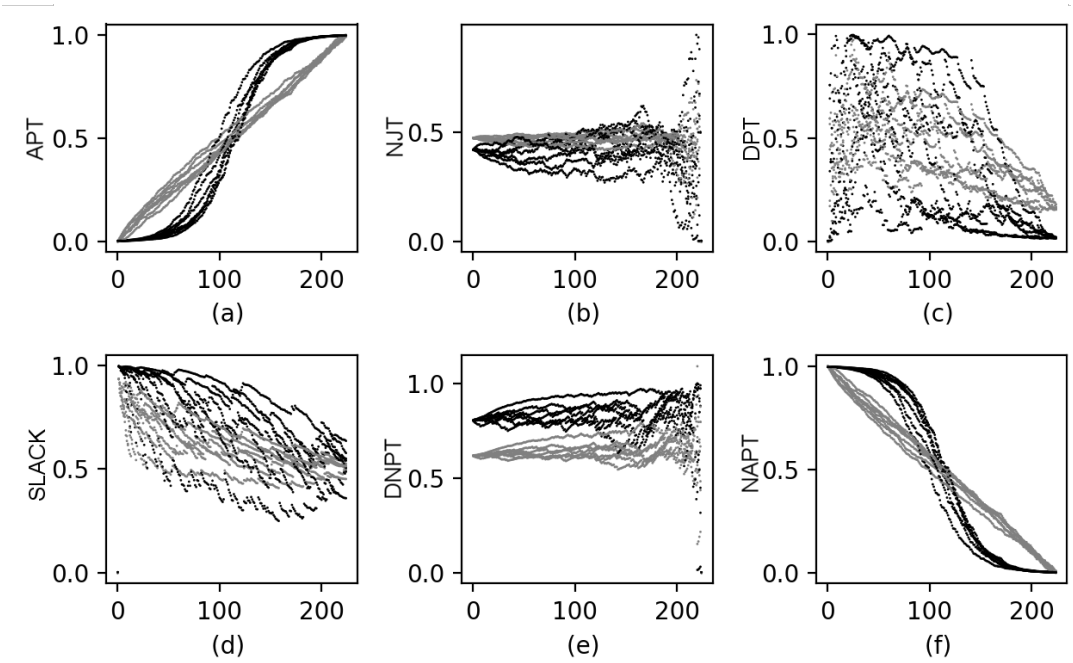

Figure 7.2: Original (gray) and ST transformed (black) feature values for all heuristics when solving an instance of size 15×15. Horizontal axis of each sub-plot is the step. Features: (a) APT, (b) NJT, (c) DPT, (d) SLACK, (e) DNPT, and (d) NAPT.

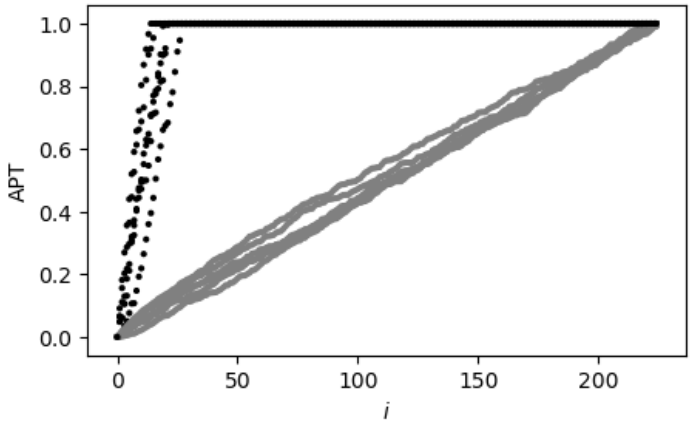

Figure 7.4: Original (gray) and LT transformed (black) values for all heuristics at step i of the solution of an instance of size 15×15. The range of the LT is [0,0.1]. Feature: APT.

7.3 Effect of Mixing Transformations (S3-E01 - S3-E08)

Although the set of features that can be used to define the state of the problem is comprised of six features, it is convenient to select a pair of them for studying their interactions. In particular, the aim is to assess the effect of assigning a different transformation to each feature, and the effect of using different transformation ranges for each feature.

It was chosen to make this study with features NJT and DNPT, since they relate to the state of the problem and not of the solution, hence a controlled version of the problem is manageable in terms of the experiments that are needed for this section. If all features were taken into account, the number of combinations to study the effects would be significant.

The expected effect of transformations is to provoke a "zooming" effect on the feature values, as was shown in Figure 7.1. Based on the behavior of those features across instances. For example, consider subfigures (b) and (e) from Figures 7.2 and 7.3. In subfigure (b), feature values fell between 0.4 and 0.6 almost until the end of the solution process. It is important to remark that, although transformations could be applied from both sides (i.e. from 0.0 to 0.4 and from 0.7 to 1.0), the range [0,0.3] was selected for the transformations. Later on, a method will be devised to adjust such parameters.

Table 7.1 presents the experiments that were planned for exploring the effect of combining transformations. As it was mentioned, the transformation range was [0,0.3] in all cases. Moreover, in all tests, the four remaining features were used in their original domain (i.e. without transformations). Each experiment was repeated 30 times. Also, 30 instances of size 5×5 were used for training and 5 instances of size 15×15 were used for testing.

Table 7.1: Experiments for analyzing the effect of combining transformations. O: no transformation. S: S-Shaped. L: Linear. In all cases, the remaining features were used in their original domain (i.e. without transformations).

Experiment	NJT	DNPT
S3-E01	O	O
S3-E02	O	L
S3-E03	L	O
S3-E04	L	L
S3-E05	L	S
S3-E06	S	L
S3-E07	S	S
S3-E08	O	S
S3-E09	S	O

Figures 7.5 and 7.6 show the resulting training and testing data, respectively. In the first case, the O-L combination seems to be the best result, but there is only a little difference in comparison with the others. For the second case, the L-O combination had the best results in terms of HH performance, but with no significant improvement for the other cases. Also, transformations seem to lower the stability of hyper-heuristics on the testing phase but also seems to increase it on the training phase. Something that can explain these results is that the noise provided by the other features (which are not being transformed) have a stronger effect than the transformed features, thus providing insights that transformations have an effect (not conclusive yet if positive or negative) on the trained HHs. The next Section describes a method to adjust the range of influence of each transformation and to select the proper transformation for the features is proposed and tested.

7.4 Tuning the Best Configuration for each Feature (S4-E09 - S4-E10)

With the results from the previous section, it is clear that choosing the best range of the transformations for each feature involves taking into account the path each feature follows throughout the solution of an instance. In this section, a method to calculate a tuned range for each feature is proposed. The way in which the method works takes into account the ideas developed in Chapter 6 about the feature dispersion that each heuristic generates when making a decision, which is illustrated in the produced paths shown in Figure 7.2. The general goal of the method is to choose the range that separates the most the feature values that each heuristic would yield at any given state, should that heuristic be applied.

The methodology to determine the feature transformation ranges involves the following steps:

1. For a problem in the training set, create a solution (without using transformations) with every available heuristic and store the feature values at each state of the solution. Note that each heuristic should generate a different solution path.

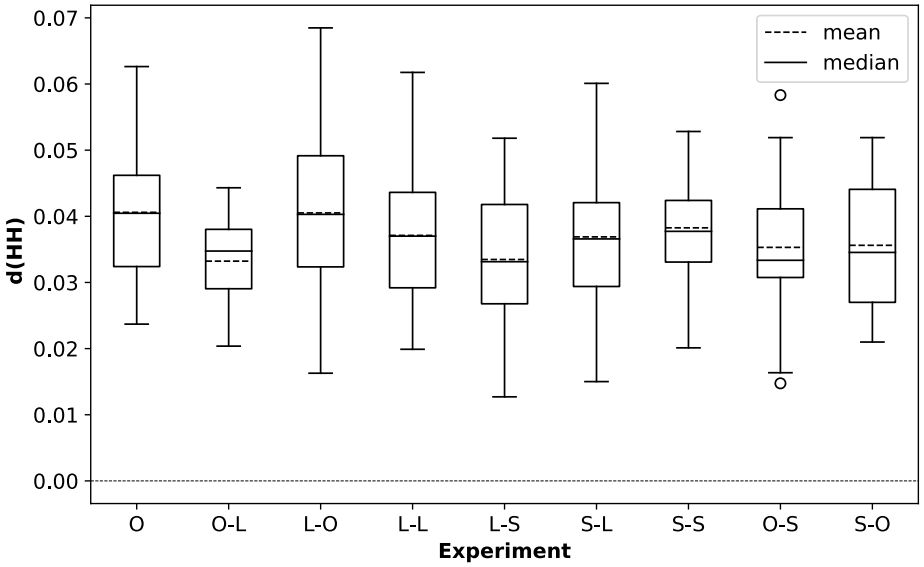

Figure 7.5: Traning results (30 runs) of experiments described in Table 7.1. $d(HH)$ given in Equation 3.2.

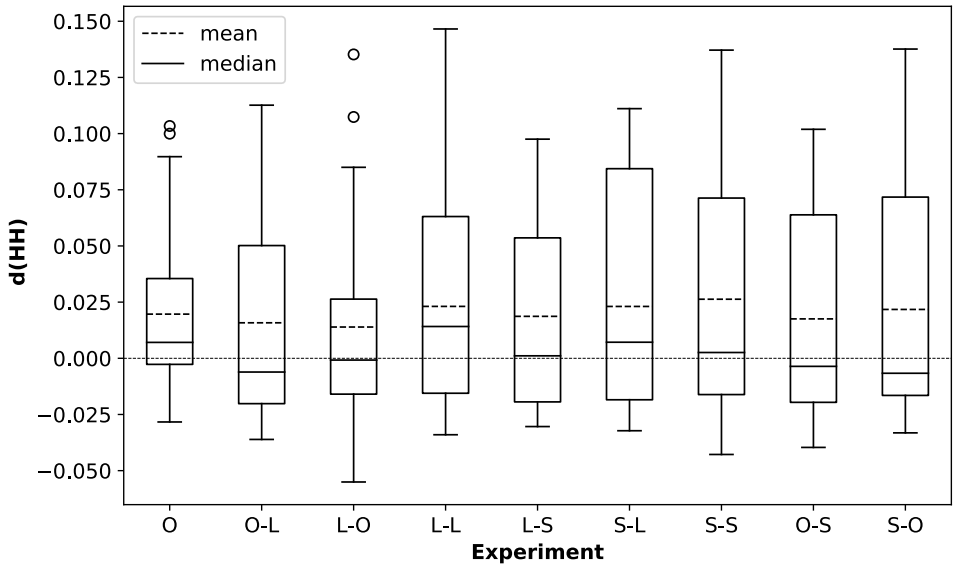

Figure 7.6: Test results (30 runs) of experiments described in Table 7.1. $d(HH)$ given in Equation 3.2.

2. Define a feature to be tuned, a number of ranges to study on the feature, and the transformation that will be used.

3. Get feature values for each solution path. For each state:

   (a) Apply the transformation to be tuned with each of the ranges defined in (2) to the feature values of every path.

   (b) For each range, measure the Euclidean distance between the original feature values and the transformed feature values.

   (c) Keep the range that produces the maximum difference between the two measures.

   This process will generate a list containing the range of maximum difference for each state.

4. For each range defined in step (2), calculate the ratio between the number of occurrences in the generated list of step (3) and the length of the solution path (this length is the same for all heuristics).

5. Return the range that has the maximum ratio calculated in (4).

6. Repeat steps (2)-(5) for each feature.

7. Repeat steps (1)-(6) for each problem in the training set. In the end, for each problem, a tuned range for each feature will be defined.

8. For each feature, take the range that occurs the most in the whole set of problems. Return the pair (feature, range).

An example of the output provided by the method for a set of instances of size 5×5 is illustrated in Table 7.2 for the LT. The same methodology can be applied to the ST or, even, to both of them. In this final case, this would allow selecting the one producing the most distance between each feature.

Table 7.2: Sample results yielded by the tuning approach when applied to the L transformation, on 30 instances of size 5×5, for the set of heuristics studied in this research. a: Lower bound of the transformation range. b: Upper bound of the transformation range.

Feature	**a**	**b**
APT	0.0	0.1
NJT	0.0	0.4
DPT	0.0	0.2
SLACK	0.7	0.9
DNPT	0.7	0.9
NAPT	0.0	0.1

To validate this approach, an experiment was devised. A training set of 30 instances with size 5×5 and a testing set of five instances with size 15×15 were defined. The experiment

consisted of comparing hyper-heuristics (trained with 1000 iterations) without transformations and hyper-heuristics generated with the aforementioned approach. Results are presented in Figure 7.7. Data confirms that hyper-heuristic training becomes more stable. In fact, the performance of O-TR varies by about 0.04, while the one for L-TR only varies in 0.02. Although not significantly different, this reinforces the idea that transformations help improve stability. In the testing phase, however, hyper-heuristics (HHs) including transformations did not outperform the original ones, at least for these conditions. This effect can be explained by

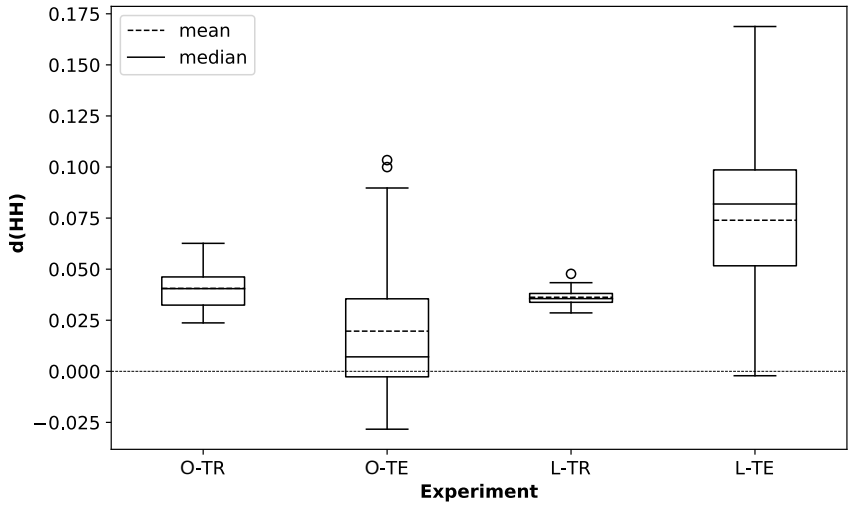

Figure 7.7: Results (30 runs) of experiments for validating the proposed approach to tune transformations. Training set: 30 instances (5×5). Testing set: 5 instances (15×15). 1000 iterations were carried out in each of the training experiments, accordingly. Statistical tests are reported in Appendix A, Table A.2.

the fact that HHs trained with transformations are able to expand feature values of the solution in such a way that the scalability (shown in Figure 6.6) is destroyed by the transformations. For the case of hyper-heuristics trained with 5×5 instances and without transformations, the generated HHs vary more in their fitness measures than those that are being transformed. Such an argument can be confirmed by changing the test set to one with the same size (5×5). Figure 7.8 shows the performance of different sets of HHs: for 100 and 1,000 iterations and with (L) and without (O) transformations over a set of 30 instances of size 5×5. In both cases, HHs that considered feature transformations outperformed the ones without them. Also, the results of L1000 tend to be skewed towards the bottom of the boxplot, while the ones of O1000 towards the center which implies that in general, the performance of HH with transformations is better. Still, in both cases, no HH was able to outperform the oracle.

7.5 Confirmatory Results (S4-E11 - S4-E12)

With the results of the previous section, it is necessary to test the effect of using feature transformations on instances of the same size, while executing the method described in Section 7.4

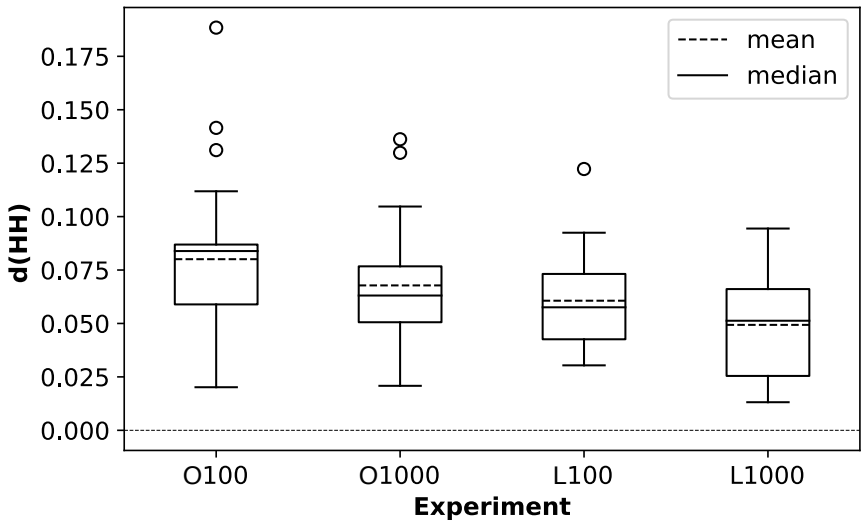

Figure 7.8: Testing performance of hyper-heuristics trained for 100 and 1000 iterations, on 30 instances of size 5×5, with and without transformations. Test set: 10 (different) instances of size 5×5. O: Hyper-heuristics without transformations. L: Hyper-heuristics with linear transformation, following the method described in Section 7.4. The number corresponds to the number of iterations done during training.

for both, S and L transformations. This allows asserting the extent to which transformations may enhance hyper-heuristic performance. Experiments with 15×15 instances were carried out because benchmark instances correspond to those published by Taillard [107], and no benchmark instances of size 5×5 were published.

A set of 30 instances of size 15×15 were generated. Then, the parameters of the transformations were tuned for all features with three configurations:

1. Select the best parameters for the linear transformation (L).
2. Select the best parameters for the S-shaped transformation (S).
3. Select the best parameters for the best (L or S) transformation for each feature.

Results of the adaptive method are presented in Table 7.3. Note that for configuration (3), the results were the same as provided for (2), hence, the experiments carried out were done by applying feature transformations with the same type of transformation for all features. 30 replicas were executed with 1000 iterations to generate HHs without transformations (O), with the S transformation (S), and with the L transformation (L). Each HH was tested on a set of 5 15×15 instances published by Taillard. Results are summarized in Figure 7.9. The effect in training is similar to the one shown for 5×5 instances. Hyper-heuristics trained with the S transformation exhibited a lower variance than those trained with the L transformation. Nevertheless, both of them outperformed the oracle. Also, they all were more stable than HHs trained without transformations.

Table 7.3: Tuning method for L, S and L/S transformations, on 30 instances of size 15×15. S: S-Shaped transformation, L: linear transformation, [a,b]: range of influence of a transformation.

Feature	S - [a, b]	L - [a, b]	S/L - [a, b]
APT	[0.0, 0.1]	[0.0, 0.1]	L - [0.0, 0.1]
NJT	[0.8, 0.9]	[0.6, 0.9]	L - [0.6, 0.9]
DPT	[0.0, 0.1]	[0.0, 0.1]	L - [0.0, 0.1]
SLACK	[0.8, 0.9]	[0.7, 0.9]	L - [0.7, 0.9]
DNPT	[0.8, 0.9]	[0.7, 0.9]	L - [0.7, 0.9]
NAPT	[0.0, 0.1]	[0.0, 0.1]	L - [0.0, 0.1]

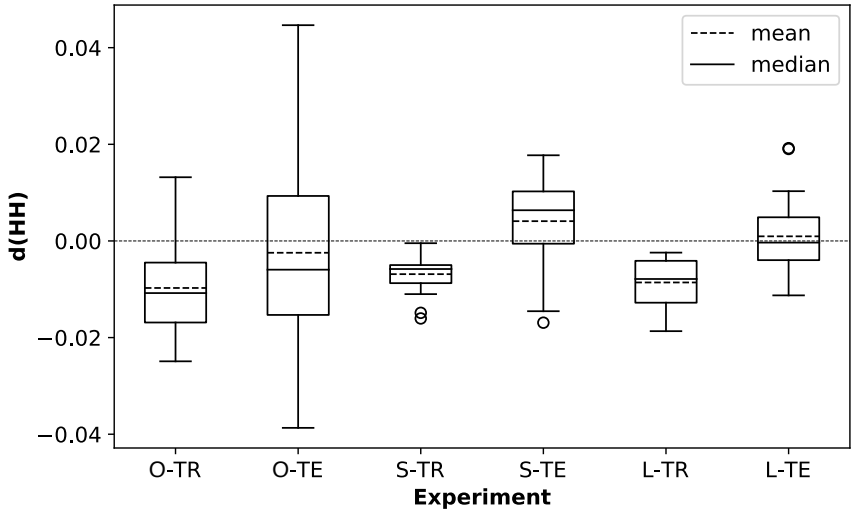

Figure 7.9: Results for HHs trained with 1000 iterations, on 30 instances of size 15×15, with and without transformations, when tested on a set of 5 benchmarking instances of the same size, proposed by Taillard. O are HHs without transformations, L with the linear transformation, S with the S transformation. TR stands for training and TE for testing. Parameters of the transformations are reported in Table 7.3. Statistical tests are reported in Appendix A, Table A.3, and results for each replica in Appendix C.

The results for the testing set also show what was previously said about transformations: Transformations yield HHs less prone to variations. On average, HHs generated with transformations delivered worse results than those without transformations. Even so, the variance is important. While HHs generated without transformations had a $d(HH)$ between -0.038 and 0.044 (8% variation), the range for HHs with S transformations (disregarding outliers) were between -0.017 to 0.017 (3.5%), while the one with L transformations ranged from -0.011 to 0.011 (3%).

7.5.1 Results with Other Benchmarking Instances (S4-E13 - S4-E14)

To conclude on the confirmation of how feature transformations enhance HHs, the generated HHs with 1,000 iterations for the cases of O (no transformation), S (S-shaped transformation) and L (linear transformation), the same HHs were tested but on a set of instances provided by authors different than Taillard [107]. The first test was carried out on instances provided by Lawrence et al. [72]. Results are shown in Figure 7.10. Both the S and L cases seem to be a little bit more stable than the ones without transformations. Also, the L HHs are slightly better than the S and O HHs. Note that in all three cases, some of the HHs were able to outperform the oracle. Here, the parameters used for the transformations were the ones shown in Table 7.3. The results confirm the stability of HHs produced with transformations but do not show significant performance improvement. This could be explained by the fact that the testing set has instances with different characteristics, hence, a way of improving the results could be to tune the transformation parameters for those instances, because the original parameters were produced with instances of Taillard.

A second test was done on 10 instances provided by Demirkol et al. [35], this time on instances of size 20×15. Results are shown in Figure 7.11. In these cases, the stability effect is notorious, but also the difference in performance. This is another confirmation that HHs generated with transformations have less variation than those produced without them, hence, the HHs without transformations can produce better results, but also much worse ones. Also, note that these are HHs that were trained with instances of size 15×15, and tested on larger (but not square) instances. This indicates that a similar scalability effect than that shown for the 5×5 case could be present, but further experiments with HHs trained on instances of the Demirkol set has to be done to confirm it.

7.6 Discussion of Results

Chapter 6 outlined the reasons why it is important to design a HH training process that takes into account the feature paths generated by available heuristics. The chapter showed that feature dispersion throughout the solution process is the main aspect of that regard. Also, Section 6.1 introduced three possible methods to achieve this and tested them accordingly.

The first one considers the size of the instance being used to train HHs. Experiments (Figure 6.3) showed that HHs trained with instances of size 5×5 for 1000 iterations achieved a better performance than those trained with 15×15 instances for 100 iterations. Although, HHs trained for 1000 iterations using instances of size 15×15 outperformed them both. Nonetheless, the median difference was small. Also, the second method considered that for a large number of iterations, the HH could be trained better. The reason: discovery of blocks that more completely describe the feature space of each state of the solution of a JSSP. Those same experiments also indicated that allowing for more iterations led to more stable and better performing HHs.

Overall, the research in Chapter 6 served to establish key points, necessary for what was studied about feature transformations. The first one was that using 5×5 instances to perform exploratory experiments was convenient and appropriate (because of the scaling effect) in terms of computational time. Although, confirmatory experiments had to be done with instances of size 15×15 to determine the extent to which feature transformations enhance HHs.

The second one was that 1000 iterations were a suitable number for studying HHs trained on instances of size 15×15. The third one was that feature transformations were promising in terms of the already described augmenting effect. This was confirmed for cases where smaller instances were used to train HHs. It is expected that feature transformations furtherly improved this effect, but for instances of other sizes.

These considerations served to study what was shown in this Chapter. At first, experiments were created to determine the extent to which the proposed transformations had any effect in separating feature values throughout the solution path. The results (Figures 7.1-7.4) indicate that, as was expected, feature transformations is useful for achieving the desired effect on feature values and heuristic paths. Section 7.3 determined that using different transformations for some of the features yielded improvements only in one of the cases (O-L). The empirically determined ranges of those experiments (Figures 7.5 and 7.6) showed a small difference in the produced HHs. But, a formal methodology for tuning those ranges was devised in Section 7.4. The idea is to maximize distances between paths of feature values produced by each heuristic. Preliminary results (Figure 7.7) with a training set of 5×5 instances showed that HHs using transformations were more stable than those without them throughout training. But, in terms of performance over the test set, transformed HHs were worse than the original ones. Since HHs were being tested on larger instances, the lower performance hinted that the effect produced by the transformations on the HHs was that the feature values (already dispersed because of the instance size) were getting much more dispersed, hence, the transformation yielded no further improvement. In fact, when testing those HHs on instances of the same size (Figure 7.8) the HHs with transformations were more stable and had better performance than the HHs without transformations.

To confirm the extent to which transformations enhance HHs, experiments were carried out with instances of size 15×15 for training them, and by using the already mentioned tuning approach. First, it was found that the L transformation performed better in all cases. In fact, when allowing the tuning procedure to choose the best transformation, the L transformation (Table 7.3) was selected in all cases. Second, the experiments showed that in general, (Figure 5.5) transformations produce more stable HHs. Even so, there are cases where HHs without transformations outperform the ones with transformations. The average performance of HHs with transformations is comparable to the one produced by HHs without them. In fact, HHs with L transformation tend to be centered around the average $d(HH)$ of original HHs. Such conclusions were coherent with the results obtained when doing tests with instances generated by other authors (Figures 7.10 and 7.11), which were discussed in Section 7.5.1.

Finally, a note is important regarding the results obtained with the transformation of only two features (Figure 7.6). When comparing the results of the training phase against the HHs trained with the same instances (5×5), but with the features transformed using the tuning method (Figure 7.7), it is clear that the latter HHs are much more stable. But, when testing them on the 15×15 instances, they performed worse than the former. This is consistent with what has been said for 5×5 instances: Transformations increase stability but at the same time lower the performance of the best-case scenario. The same happens when HHs with only two transformed features are compared against the ones trained with 15×15 instances and with all transformed features. Since the latter were trained on problems of the same size, results are as good as the other ones, but quite more stable.

Overall, what was found in this research regarding feature transformations and their potential in terms of enhancing hyper-heuristics can be summarized as follows:

- Hyper-heuristics trained with a sufficient number of iterations on small instances and tested on large instances show similar performance to those trained with larger instances when tested over the same set. This is what is called a scaling effect throughout this thesis.
- The aforementioned scaling effect is produced because features of smaller instances are more disperse. And the experiments tend to show that this dispersion is a very relevant aspect when training selection constructive HHs of the kind used throughout this thesis.
- Feature transformations have the purpose of augmenting the dispersion of feature values. It was seen that HHs generated with feature transformations are far more stable than those produced without them.
- Results indicate that feature transformations destroy the improvement derived from scaling. An explanation for this is that the dispersion of feature values is already big in smaller instances in comparison to the dispersion in larger instances. Therefore, when feature transformations are used, the HH is not able to generalize. In other words, using HH with transformations is only feasible on instances of the same size used for training.
- When transforming only a subset of features, some of the generalization capability is preserved. It seems that, with a proper combination of original and tuned transformed features, HHs could perform even better than those which exhibit the scaling effect.
- Adjusting the ranges by maximizing the distance of feature values between different solution paths produced by a single heuristic yield more stable HHs in training and testing as long as both sets are of the same size. Such HHs perform as well as the average $d(HH)$ of HHs generated without transformation, but with the advantage that they are considerably more stable.

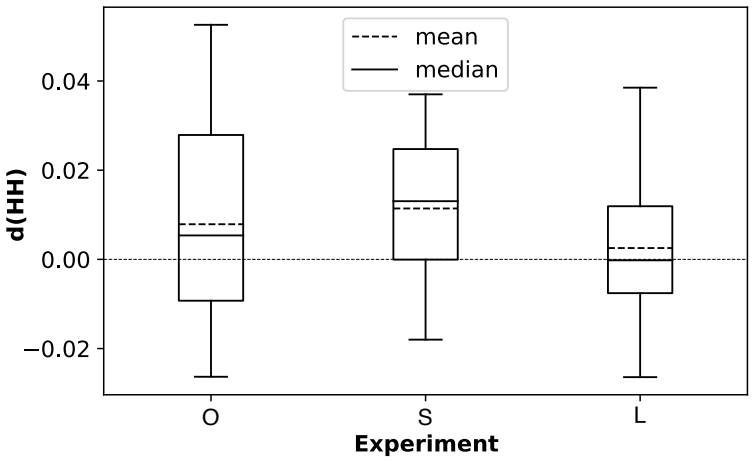

Figure 7.10: Results for 30 HHs trained with 1000 iterations, on 10 instances of size 15×15, with and without transformations, when tested on a set of 10 benchmarking instances of size 15×15, proposed by Lawrence et al. [72]. O: HHs without transformations; L: HHs linear transformation; S: HHs with S-shaped transformation. Parameters of the transformations are reported in Table 7.3.

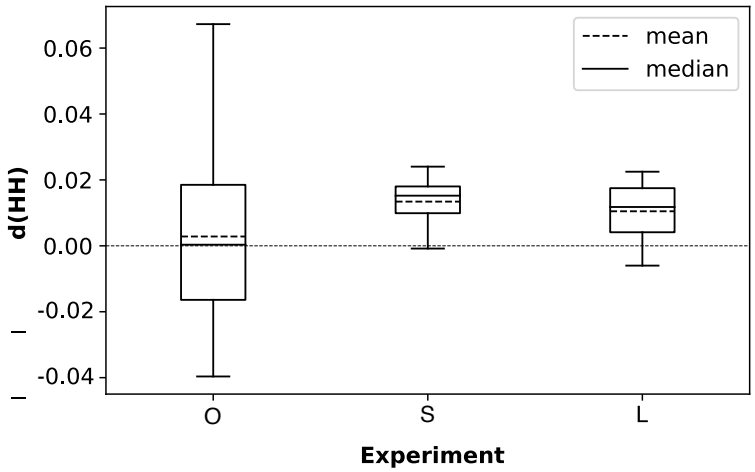

Figure 7.11: Results for HHs trained with 1000 iterations, on 10 instances of size 15×15, with and without transformations, when tested on a set of 10 benchmarking instances of size 20×15, proposed by Demirkol et al. [35]. O: HHs without transformations; L: HHs linear transformation; S: HHs with S-shaped transformation. Parameters of the transformations are reported in Table 7.3.

7.7 Summary

The goal of this Chapter was to englobe all the findings so far provided by the research and take them into account to assert the extent to which feature transformations enhance selection constructive hyper-heuristics. In this manner, two main effects were expected of feature transformations: first, that they should be able to reduce the likeliness of solution paths covered by each heuristic, and second, that they should enhance the feature space, in terms of augmenting dispersion between feature values. These effects are furtherly studied and illustrated with preliminary experiments. It is shown that the feature transformations proposed were able to produce more distinct solution paths and feature values during the process of construction of a schedule. Also, the Chapter presents that some positive effects achieved by the scaling effect shown in Chapter 6 are furtherly destroyed by feature transformations, except for the case where not all features were transformed.

A method to tune the ranges of influence of the transformations is also described and implemented. It was shown that HHs generated with the parameters provided by this method for the transformations are very stable when compared to those HHs generated without transformations, and for the case when HHs are trained and tested in instances of the same size, the former HHs are also better in terms of performance. Also, in all cases, the linear transformation seems to perform better than the S-shaped one.

Finally, a discussion with the aggregate findings of the Chapter is presented, by taking into account the overall findings of the thesis, and concludes with a summary of the conclusions regarding how feature transformations enhance hyper-heuristics.

Chapter 8

Conclusions

This thesis focused on two goals: Developing and using hyper-heuristics (HHs) for JSSPs and studying the extent to which feature transformations enhance their performance. In the first part, the study centered on analyzing the performance of HHs with a predefined set of features. In the second one, the study determined the main aspects of feature transformations that need to be taken into account when using them to train hyper-heuristics, and how they impact performance. To achieve these goals, a model of selection HH and feature transformations was proposed and tested on various problem instances, measuring performance by comparing them to the heuristics that achieve the best results for a given instance, i.e. the Oracle.

This chapter describes the main achievements of the research, according to the objectives that were stated in Section 1.4. The primary conclusions from the experiments are also derived. Afterward, conclusions are discussed and their generalization is explained. The main issues related to the proposed model, as well as the main insights from this research, are also commented on. Finally, the chapter wraps up with an outline of future research paths.

8.1 Achievements

The thesis proposed a model for training constructive selection HHs through Simulated Annealing (Chapter 3). The model was tested on classic Job Shop instances published in the literature. Preliminary experiments (Chapter 5) showed that for the same size of instances, and while keeping iterations to a small number, HHs achieve similar performances in both, training and testing. Although some of the HHs performed better than the oracle, the average performance was 5% worse than the oracle. Then, a study on the effect of impact size was outlined (Chapter 6. A hypothesis with several considerations for such an effect was also presented. Two of them were addressed within the chapter, by studying the effect of training with small instances while testing on bigger ones. Experiments showed that HHs trained with 5×5 instances for 1000 iterations had a better performance on 15×15 instances than those trained with instances of the same size for 100 iterations. Besides the performance gain, HHs generated with smaller instances took considerable less time to train than those trained with 15×15. Chapter 7 focused on the extent to which feature transformations enhance HH training. A method to adjust the parameters of two transformations was described. Results indicate that while transformations help to produce more stable HHs, performance was not

necessarily enhanced.

8.2 Main Conclusions

This section presents the conclusions for the research objectives of the thesis. Conclusions are outlined from results obtained in Chapters 5-7.

8.2.1 Heuristics Chosen and Hyper-heuristic Potential

Prior to analyzing how HHs perform on the JSSP, an analysis about the selected heuristics was carried out by comparing them to an approximate lower bound of the instance. Such an analysis was presented in Chapter 5. Experiments were done for 5 sets of 100 instances of different sizes, and results show that the selected heuristics perform better in smaller instances than in larger ones. Also, heuristics EST and MRT were found to be considerably better than the others, with the latter showing best performance in all sets of instances. Besides, a study about path similarity when building a solution for 5×5 and 15×15 instances revealed that approximately 15%-18% of the heuristics choose the same action at the beginning and at the latter steps of the solutions. But, throughout the solution process, this metric was around 5% for instances of small size, and it diminished for larger instances. This trend indicates that combining different heuristics produces different results than when using a single heuristic. To assert whether this solution can be better, an experiment with a random hyper-heuristic was carried out on 100 instances for both sizes. Results indicate that over 15% of the generated HHs produce better results than using a single heuristic. Nevertheless, the difference is small in comparison to the Oracle.

8.2.2 Hyper-heuristic Behavior in the JSSP

The objective of hyper-heuristics is to produce a solution that is at least as good as the Oracle. Chapter 5 presented an analysis of the effect the number of training iterations have on HH performance. Results indicate that, in general, training with 1,000 iterations yields HHs with stable performance. Also, training for just 100 iterations leads to HHs with a higher variance. The benefit of doing so is that they can sometimes outperform the oracle. The tendency in instances of size 5×5 was that the generalization capacity diminished as more instances were added to the test set. Because of this, Chapter 6 presented experiments to assess the impact of instance size and scaling. Results indicate that HHs trained with instances of size 5×5 and 1000 iterations have better performance on 15×15 instances than HHs trained with instances of the same size, and that is also generated with considerably less computational time. Reasons for such effect were further outlined and confirmed. It seems that the smaller the instance, the higher the variety of heuristics for properly solving it. Moreover, HHs performed better with smaller instances because feature values were more dispersed than in larger ones. This conclusion paved the way towards using feature transformations.

8.2.3 Feature Transformations

Chapter 7 studied feature transformations, following the argument that achieving maximum separation of feature values throughout the solution process could enhance HH performance. Specifically, linear and S-shaped transformations were tackled in this work. Features DNPT and NJT where chosen since they describe the state of the problem, and this allows detecting an effect without having to solve the instance. It was shown that by applying a transformation, the feature values were more separated between them. After this, it was shown that transformation ranges needed to be adjusted because choosing an empirical value did not yield noticeable results. A method for determining the values for the transformations was proposed, implemented and tested. Results indicated that by using the method to determine the ranges and applying feature transformations, the HHs generated were considerably more stable than the HHs generated without transformations. Also, the linear transformation yielded better results than the S-shaped transformation. Nevertheless, this did not lead to better performing HHs. In fact, for the cases of instances 15×15, the performance of the HHs with transformations was similar to the average of those generated without transformations. It was confirmed that the proposed approach yields more stable HHs. So, the idea of maximizing the distance between the feature values (Chapter 6) was confirmed. Finally, it is important to note that in all cases, results are still far from the approximated theoretical lower bound for the instances.

8.3 Discussion

The method to generate hyper-heuristics that was proposed, implemented and tested in this thesis is novel in the Job Shop (JS) domain. Hence, this research can be seen as a first approach to solve the JS problem by applying dispatching rules in a dynamic way throughout the construction of the schedule. Results are promising, but there are several issues of note.

The set of chosen heuristics is a major limitation for this research. The set is comprised of only six rules, and two of them (EST and MRT) are much better in terms of performance than the remaining four. From the experiments, it was shown that combining rules could lead to better results than using them by themselves. However, generated HHs are still far from the optimum solution. It seems that a broader set of rules need to be tested if the objective is reaching an optimal solution, or if it is to study the performance of the proposed model more robustly.

This research is also limited to instances of sizes 5×5 and 15×15. Also, the number of iterations used in this research greatly restricted the number of experiments that could be done, because of restrictions in available computational resources. Several optimizations can be done to the framework to generate results faster. Thus, it would be able to produce data for more and larger instances.

Likewise, the approach considered to train HHs is questionable. It is possible that by using a different method, or by modifying SA, better results can be achieved. The reason for this is that some of the generated blocks may not be applicable to any state during the construction of the solution. Hence, it would be very difficult for the meta-heuristic to find better solutions.

The scaling effect (Chapter 6) was destroyed by the results presented in Chapter 7 about feature transformations. It seems that when feature transformations are applied, the scaling

effect disappears. One reason for this could be that the feature space is redistributed following the transformation, so regions no longer correspond. Still, data about the scaling effect were important, because they allowed producing an argument and theory on which to base the reasons to apply feature transformations. The latter shows that HHs generated with this process are very stable. Hence, although further research needs to be done to improve their performance, this stands as a contribution of the research for developing HHs with better generalization capabilities than the ones produced without transformations. Nevertheless, this is still an open topic of research that needs further research, not only to produce results that are better than an oracle but also comparable to those produced by mathematical optimizers.

8.4 Future Work

Using HHs like the ones proposed in this thesis for solving JSS problems is still an area that needs further research, and several ideas arose throughout this project. This section outlines such ideas and presents some ways in which they could be implemented.

8.4.1 Enlarging the Set of Heuristics

As it was described in Chapter 2 and discussed in the past section, there is a large set of dispatching rules in the literature, which could be used to generate HHs. Some potential work that could be done in this matter is: (1) Implement a large set of dispatching rules in the framework used for this thesis. (2) Study their performance by comparing them to the lower bound of a large set of instances, such as the study done in Section 5.1. (3) Rank them and choose a broader set of rules which have the best performance. (4) Use the selected rules to train HHs with the same model proposed in this thesis.

8.4.2 Feature Selection

The features used in this research were selected arbitrarily. The idea was to select features that could describe the state of both, the problem and the solution. Feature selection is an important approach in Machine Learning that may improve HH performance. However, doing so was outside the scope of this work. Nonetheless, using a broader set of features and doing a proper analysis could be an interesting path for future work. A possible study on features could be to select a set of features and study them in relation to the diversity of the values they produce when a specific heuristic is being applied. Also, the training process of HHs could be restricted to the granular differences in the values those features have throughout the solution process. This way, the generated HHs will not produce blocks with a combination of feature values that will be scarcely present throughout the solution of an instance. In addition, as it was pointed out in Section 3.2, a methodology similar to the one proposed by Mirshekarian et al. [85] to study the set of features before applying the prior analysis could be followed.

8.4.3 Hybrid SA-Tabu Search

Since HHs generate blocks in a stochastic way, the search process is computationally inefficient. Better results may be obtained by creating a list of previously generated blocks of features-actions that are never used when solving the training instances. Such a list could be used as a restriction for generating blocks during the SA training process.

8.4.4 Framework Optimization

This thesis contributes with a framework for generating HHs for solving the JSSP. Nevertheless, since this is a first approach, further coding optimizations could be done to reduce the computational time the framework needs. During a profiling made on the framework, it was seen that one of the stages that takes the longest time is producing a feasible activity/time pair. This set is used by the heuristics to select the activity to schedule next. It seems that a possible optimization is to rewrite parts of the code to make vectorized versions of the methods, which are more efficient than traditional list comprehensions or for loops. Also, only one part of the code was run in parallel. There are other parts of the framework where the use of parallel computation could be of benefit.

8.4.5 Benchmarking

Although the focus of this research was not to compare the technique proposed against state-of-the-art algorithms, it is desirable to perform experiments by taking instances published in the literature and compare the makespan achieved by this method against the published ones for other algorithms such as TSAB [89]. Nevertheless, for illustration purposes, results for the confirmatory experiments (over 5 of Taillard's 15×15 instances) are reported in Table C.1, so as to serve as a point of comparison if the aforementioned experiments are carried out.

8.4.6 Hyper-heuristics of hyper-heuristics

During the evolution of this thesis research, it was found that hyper-heuristics trained with feature transformations are more stable than those produced without them. Several experiments done with one instance for training indicate that the generated HHs were, at least, as good as the Oracle. In many cases, they were actually much better than it. This leads to two research paths: (1) For industry problems, a HH with very good performance can be generated, given a broad set of rules, which is at least as good as the best single heuristic; (2) Hyper-heuristics generated with feature transformations with single instances for training are very stable. So, one could produce highly specialized hyper-heuristics and then train a higher-level hyper-heuristic that selects among them. To detect whether hyper-heuristics of hyper-heuristics produce better results also seems an interesting line of research.

Appendix A

Statistical Tests

Table A.1: Wilcoxon's signed rank sum test over the results from experiments of S2 (Figure 6.6), at $\alpha = 0.05$. The result is either Same or Different meaning the test implies that the experiments come from the same or from different distribution.

Experiment	Stat	p-value	Result
55-1000-TE / 1515-100-TE	360.5	0.094	Same
55-1000-TR / 1515-100-TR	329	0.037	Different
55-1000-TE / 1515-1000-TE	291	0.01	Different

Table A.2: Wilcoxon's signed rank sum test over the results from experiments of S4 (Figure 7.7), at $\alpha = 0.05$. The result is either Same or Different meaning the test implies that the experiments come from the same or from different distribution.

Experiment	Stat	p-value	Result
O-TR / L-TR	267	0.003	Different
O-TE / L-TE	135	0.000	Different

Table A.3: Wilcoxon's signed rank sum test over the results from experiments of S4 (Figure 7.9), at $\alpha = 0.05$. The result is either Same or Different meaning the test implies that the experiments come from the same or from different distribution.

Experiment	**Stat**	**p-value**	**Result**
O-TR / S-TR	314	0.0225	Different
O-TR / L-TR	385	0.170	Same
S-TR / L-TR	391.5	0.195	Same
O-TE / S-TE	308	0.0182	Different
O-TE / L-TE	351	0.072	Same
S-TE / L-TE	330.5	0.039	Different

Appendix B

Effect of the Number of Iterations During Training

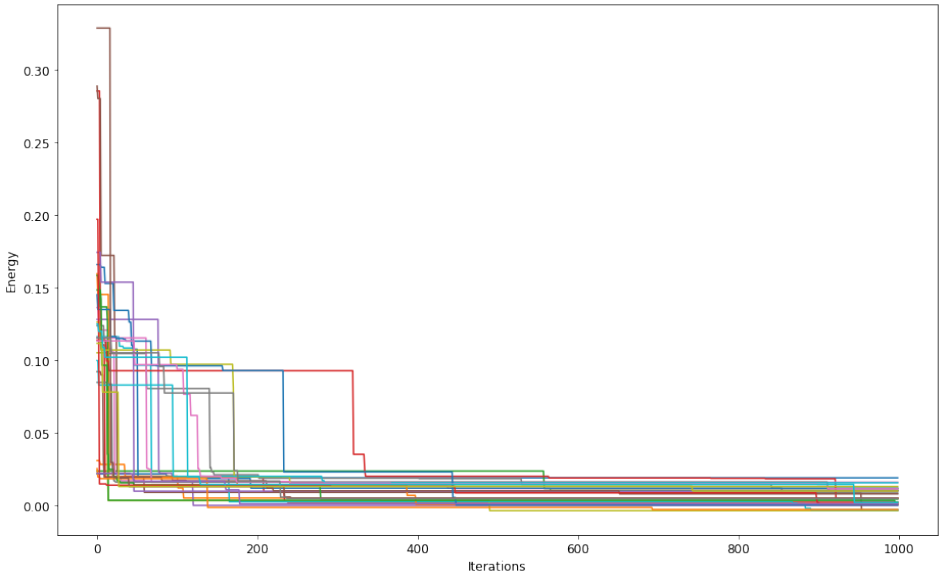

Figure B.1: Illustration of how energy diminishes throughout the number of iterations when applying the training algorithm. Each line represents a different HH trained with 30 instances of size 15×15. In total, 30 replicas are shown.

Appendix C

Confirmatory Experiments Results

Table C.1: Results of confirmatory experiments of S4 (Figure 7.3) for 30 replicas. Trained without transformations (Original), with Linear (LT) and S-Shaped (ST) transformations, with 30 instances of size 15×15 and tested over the 5 benchmarking instances of size 15×15.

Original	**ST**	**LT**
0.020108032	0.009311475	0.019036951
0.00392605	0.01699375	0.010300349
-0.003163924	0.005822668	0.001351308
-0.016230976	0.00655425	-0.004521326
0.044654531	0.010300349	-0.000576742
-0.018933261	0.006826882	0.006102818
-0.020161947	-0.000576742	-0.000103292
0.013307642	0.011826252	-0.001092334
0.010872921	-0.010165683	0.005008191
0.013307642	-0.006225365	-0.004625515
0.032469275	0.009226993	-0.011262946
0.009311475	0.017729892	0.019199466
-0.012556215	0.001350257	-0.00790682
-0.025551279	0.01545988	-0.005384562
-0.018077313	-0.013826647	-0.007712109
-0.007227476	0.015764242	0.009088321
-0.018307935	0.015167609	0.004565342
-0.005702126	0.010113092	0.006850859
-0.011060024	0.00635846	-0.000576742
-0.006189271	0.006352115	-0.001937178
0.000488366	0.002137821	-0.003533541
-0.019178856	0.007283956	0.002127348
0.019798376	-0.002651021	8.24E-05
-0.007564502	-0.014540697	-0.003932357
-0.008964928	-0.000504515	-8.74E-05
0.009311475	-0.016915963	0.004565342
-0.000955361	0.003898311	0.008387132
-0.000928194	-0.006024358	-0.001713573
-0.011513138	0.016312584	-0.009037846
-0.038684105	-0.000546481	-0.003987012

Bibliography

[1] ABDOLRAZZAGH-NEZHAD, M., AND ABDULLAH, S. Job shop scheduling: Classification, constraints and objective functions. International Journal of Computer and Information Engineering 11, 4 (2017), 429–434.

[2] ADAMS, J., BALAS, E., AND ZAWACK, D. The shifting bottleneck procedure for job shop scheduling. Management Science 34, 3 (1988), 391–401.

[3] AMAYA, I., ORTIZ-BAYLISS, J. C., GUTIÉRREZ-RODRÍGUEZ, A. E., TERASHIMA-MARÍN, H., AND COELLO, C. A. C. Improving hyper-heuristic performance through feature transformation. In 2017 IEEE Congress on Evolutionary Computation (CEC) (June 2017), pp. 2614–2621.

[4] AMAYA, I., ORTIZ-BAYLISS, J. C., ROSALES-PEREZ, A., GUTIERREZ-RODRIGUEZ, A. E., CONANT-PABLOS, S. E., TERASHIMA-MARÍN, H., AND COELLO, C. A. C. Enhancing selection hyper-heuristics via feature transformations. IEEE Computational Intelligence Magazine 13, 2 (May 2018), 30–41.

[5] APPLEGATE, D., AND COOK, W. A computational study of the job-shop scheduling problem. ORSA Journal on Computing 3, 2 (1991), 149–156.

[6] ARTIGUES, C., AND FEILLET, D. A branch and bound method for the job-shop problem with sequence-dependent setup times. Annals of Operations Research 159, 1 (2008), 135–159.

[7] BAI, R., BLAZEWICZ, J., BURKE, E. K., KENDALL, G., AND MCCOLLUM, B. A simulated annealing hyper-heuristic methodology for flexible decision support. 4OR 10, 1 (Mar 2012), 43–66.

[8] BALAS, E. An additive algorithm for solving linear programs with zero-one variables. Operations Research 13, 4 (1965), 517–546.

[9] BALAS, E. Discrete programming by the filter method. Operations Research 15, 5 (1967), 915–957.

[10] BALAS, E. Machine sequencing via disjunctive graphs: An implicit enumeration algorithm. Operations Research 17, 6 (1969), 941–957.

[11] BAPTISTE, P., LE PAPE, C., AND NUIJTEN, W. Constraint-based scheduling: applying constraint programming to scheduling problems, vol. 39. Springer Science & Business Media, 2012.

[12] BAYKASOĞLU, A., AND KARASLAN, F. S. Solving comprehensive dynamic job shop scheduling problem by using a grasp-based approach. International Journal of Production Research 55, 11 (2017), 3308–3325.

[13] BECK, J. C., FENG, T., AND WATSON, J.-P. Combining constraint programming and local search for job-shop scheduling. INFORMS Journal on Computing 23, 1 (2011), 1–14.

[14] BILGIN, B., ÖZCAN, E., AND KORKMAZ, E. E. An experimental study on hyper-heuristics and exam timetabling. In International Conference on the Practice and Theory of Automated Timetabling (2006), Springer, pp. 394–412.

[15] BLUM, C., AND SAMPELS, M. An ant colony optimization algorithm for shop scheduling problems. Journal of Mathematical Modelling and Algorithms 3, 3 (Sep 2004), 285–308.

[16] BOWMAN, E. H. The schedule-sequencing problem. Operations Research 7, 5 (1959), 621–624.

[17] BRATLEY, P., FOX, B. L., AND SCHRAGE. A guide to simulation. Springer Verlag, 1983.

[18] BRUCKER, P. A polynomial algorithm for the two machine job-shop scheduling problem with a fixed number of jobs. Operations-Research-Spektrum 16, 1 (Mar 1994), 5–7.

[19] BURKE, E. K., GENDREAU, M., HYDE, M., KENDALL, G., OCHOA, G., ÖZCAN, E., AND QU, R. Hyper-heuristics: A survey of the state of the art. Journal of the Operational Research Society 64, 12 (2013), 1695–1724.

[20] BURKE, E. K., HYDE, M., KENDALL, G., OCHOA, G., ÖZCAN, E., AND WOODWARD, J. R. A classification of hyper-heuristic approaches. In Handbook of metaheuristics. Springer, 2010, pp. 449–468.

[21] BURKE, E. K., HYDE, M. R., KENDALL, G., OCHOA, G., ÖZCAN, E., AND WOODWARD, J. R. A classification of hyper-heuristic approaches: Revisited. In Handbook of Metaheuristics. Springer, 2019, pp. 453–477.

[22] BURKE, E. K., KENDALL, G., AND SOUBEIGA, E. A tabu-search hyperheuristic for timetabling and rostering. Journal of heuristics 9, 6 (2003), 451–470.

[23] BURKE, E. K., MCCOLLUM, B., MEISELS, A., PETROVIC, S., AND QU, R. A graph-based hyper-heuristic for educational timetabling problems. European Journal of Operational Research 176, 1 (2007), 177–192.

[24] BŁAŻEWICZ, J., DOMSCHKE, W., AND PESCH, E. The job shop scheduling problem: Conventional and new solution techniques. European Journal of Operational Research 93, 1 (1996), 1 – 33.

[25] CARLIER, J., AND PINSON, E. An algorithm for solving the job-shop problem. Management Science 35, 2 (1989), 164–176.

[26] CHAKHLEVITCH, K., AND COWLING, P. Hyperheuristics: recent developments. In Adaptive and multilevel metaheuristics. Springer, 2008, pp. 3–29.

[27] CHAURASIA, S. N., SUNDAR, S., JUNG, D., LEE, H. M., AND KIM, J. H. An evolutionary algorithm based hyper-heuristic for the job-shop scheduling problem with no-wait constraint. In Harmony Search and Nature Inspired Optimization Algorithms (Singapore, 2019), N. Yadav, A. Yadav, J. C. Bansal, K. Deep, and J. H. Kim, Eds., Springer Singapore, pp. 249–257.

[28] CHEN, H., AND LUH, P. B. An alternative framework to lagrangian relaxation approach for job shop scheduling. European Journal of Operational Research 149, 3 (2003), 499–512.

[29] CHENG, R., GEN, M., AND TSUJIMURA, Y. A tutorial survey of job-shop scheduling problems using genetic algorithms—i. representation. Computers & industrial engineering 30, 4 (1996), 983–997.

[30] CHENG, T. C., E., PENG, B., AND LÜ, Z. A hybrid evolutionary algorithm to solve the job shop scheduling problem. Annals of Operations Research 242, 2 (07 2016), 223–237.

[31] CHEUNG, W., AND ZHOU, H. Using genetic algorithms and heuristics for job shop scheduling with sequence-dependent setup times. Annals of Operations Research 107, 1-4 (2001), 65–81.

[32] COWLING, P., KENDALL, G., AND SOUBEIGA, E. A hyperheuristic approach to scheduling a sales summit. In International Conference on the Practice and Theory of Automated Timetabling (2000), Springer, pp. 176–190.

[33] DANACH, K. Hyperheuristics in Logistics. PhD thesis, Ecole Centrale de Lille, 2016.

[34] DAVIS, L. Job shop scheduling with genetic algorithms. In Proceedings of an international conference on genetic algorithms and their applications (1985), vol. 140.

[35] DEMIRKOL, E., MEHTA, S., AND UZSOY, R. A computational study of shifting bottleneck procedures for shop scheduling problems. Journal of Heuristics 3, 2 (1997), 111–137.

[36] DEMIRKOL, E., MEHTA, S., AND UZSOY, R. Benchmarks for shop scheduling problems. European Journal of Operational Research 109, 1 (1998), 137 – 141.

[37] DOWSLAND, K. A., SOUBEIGA, E., AND BURKE, E. A simulated annealing based hyperheuristic for determining shipper sizes for storage and transportation. European Journal of Operational Research 179, 3 (2007), 759–774.

[38] DUBOIS, D., FARGIER, H., AND PRADE, H. Fuzzy constraints in job-shop scheduling. Journal of Intelligent Manufacturing 6, 4 (Aug 1995), 215–234.

[39] DUECK, G., AND SCHEUER, T. Threshold accepting: A general purpose optimization algorithm appearing superior to simulated annealing. Journal of Computational Physics 90, 1 (1990), 161 – 175.

[40] EL-BOURI, A., AND SHAH, P. A neural network for dispatching rule selection in a job shop. The International Journal of Advanced Manufacturing Technology 31, 3-4 (2006), 342–349.

[41] FACCIO, M., NEDAEI, M., AND PILATI, F. A new approach for performance assessment of parallel and non-bottleneck machines in a dynamic job shop environment. International Journal of Energy Sector Management (2019).

[42] FISHER, M. L. Optimal solution of scheduling problems using lagrange multipliers: Part i. Operations Research 21, 5 (1973), 1114–1127.

[43] FISHER, M. L. Optimal solution of scheduling problems using lagrange multipliers: Part ii. In Symposium on the Theory of Scheduling and its Applications (1973), Springer, pp. 294–318.

[44] FRANZIN, A., AND STÜTZLE, T. Revisiting simulated annealing: A component-based analysis. Computers & Operations Research 104 (2019), 191–206.

[45] GAO, J., SUN, L., AND GEN, M. A hybrid genetic and variable neighborhood descent algorithm for flexible job shop scheduling problems. Computers & Operations Research 35, 9 (2008), 2892–2907.

[46] GAO, L., LI, X., WEN, X., LU, C., AND WEN, F. A hybrid algorithm based on a new neighborhood structure evaluation method for job shop scheduling problem. Computers & Industrial Engineering 88 (2015), 417 – 429.

[47] GARCÍA, S., LUENGO, J., AND HERRERA, F. Data preprocessing in data mining. Springer, 2015.

[48] GAREY, M. R., JOHNSON, D. S., AND SETHI, R. The complexity of flowshop and jobshop scheduling. Mathematics of operations research 1, 2 (1976), 117–129.

[49] GEMAN, S., AND GEMAN, D. Stochastic relaxation, gibbs distributions, and the bayesian restoration of images. In Readings in computer vision. Elsevier, 1987, pp. 564–584.

[50] GERE JR, W. S. Heuristics in job shop scheduling. Management Science 13, 3 (1966), 167–190.

[51] GHAFFARINASAB, N., MOTALLEBZADEH, A., JABARZADEH, Y., AND KARA, B. Y. Efficient simulated annealing based solution approaches to the competitive single and multiple allocation hub location problems. Computers & Operations Research 90 (2018), 173–192.

[52] GOMEZ, J. C., AND TERASHIMA-MARÍN, H. Evolutionary hyper-heuristics for tackling bi-objective 2D bin packing problems. Genetic Programming and Evolvable Machines 19, 1-2 (2018), 151–181.

[53] GONÇALVES, J. F., DE MAGALHÃES MENDES, J. J., AND RESENDE, M. G. A hybrid genetic algorithm for the job shop scheduling problem. European journal of operational research 167, 1 (2005), 77–95.

[54] GRAHAM, R., LAWLER, E., LENSTRA, J., AND KAN, A. Optimization and approximation in deterministic sequencing and scheduling: a survey. In Discrete Optimization II, P. Hammer, E. Johnson, and B. Korte, Eds., vol. 5 of Annals of Discrete Mathematics. Elsevier, 1979, pp. 287 – 326.

[55] H. BLACKSTONE, J., T. PHILLIPS, D., AND HOGG, G. A state-of-the-art survey of dispatching rules for manufacturing job shop operations. International Journal of Production Research 20 (01 1982), 27–45.

[56] HART, E., SIM, K., GARDINER, B., AND KAMIMURA, K. A hybrid method for feature construction and selection to improve wind-damage prediction in the forestry sector. In Proceedings of the Genetic and Evolutionary Computation Conference (2017), ACM, pp. 1121–1128.

[57] HAUPT, R. A survey of priority rule-based scheduling. Operations-Research-Spektrum 11, 1 (1989), 3–16.

[58] HERRMANN, J. W. Handbook of production scheduling, vol. 89. Springer Science & Business Media, 2006.

[59] JAIN, A. S., AND MEERAN, S. Job-shop scheduling using neural networks. International Journal of production research 36, 5 (1998), 1249–1272.

[60] JAIN, A. S., AND MEERAN, S. A state-of-the-art review of job-shop scheduling techniques. Tech. rep., Technical report, Department of Applied Physics, Electronic and Mechanical Engineering, 1998.

[61] JUN, S., LEE, S., AND CHUN, H. Learning dispatching rules using random forest in flexible job shop scheduling problems. International Journal of Production Research (2019), 1–21.

[62] KALENDER, M., KHEIRI, A., ÖZCAN, E., AND BURKE, E. K. A greedy gradient-simulated annealing selection hyper-heuristic. Soft Computing 17, 12 (2013), 2279–2292.

[63] KARAGUL, K., SAHIN, Y., AYDEMIR, E., AND ORAL, A. A simulated annealing algorithm based solution method for a green vehicle routing problem with fuel consumption. In Lean and Green Supply Chain Management. Springer, 2019, pp. 161–187.

[64] KHEIRI, A., AND KEEDWELL, E. A sequence-based selection hyper-heuristic utilising a hidden markov model. In Proceedings of the 2015 Annual Conference on Genetic and Evolutionary Computation (2015), ACM, pp. 417–424.

[65] KIRKPATRICK, S., GELATT, C. D., AND VECCHI, M. P. Optimization by simulated annealing. Science 220, 4598 (1983), 671–680.

[66] KONDILI, E., PANTELIDES, C., SARGENT, R., ET AL. A general algorithm for scheduling batch operations. In PSE'88: Third International Symposium on Process Systems Engineering: In Affiliation with CHEMECA 88, a Bicentennial Event; Sydney, Australia, 28 August-2 September, 1998; Preprints of Papers (1988), Institution of Engineers, Australia, p. 62.

[67] KU, W.-Y., AND BECK, J. C. Mixed integer programming models for job shop scheduling: A computational analysis. Computers & Operations Research 73 (2016), 165–173.

[68] KUNDAKCI, N., AND KULAK, O. Hybrid genetic algorithms for minimizing makespan in dynamic job shop scheduling problem. Computers & Industrial Engineering 96 (2016), 31–51.

[69] KURDI, M. An effective new island model genetic algorithm for job shop scheduling problem. Computers & Operations Research 67 (2016), 132 – 142.

[70] LAGEWEG, B. J., LENSTRA, J. K., AND KAN, A. H. G. R. Job-shop scheduling by implicit enumeration. Management Science 24, 4 (1977), 441–450.

[71] LAGUNA, M., AND MARTI, R. Scatter search: methodology and implementations in C, vol. 24. Springer Science & Business Media, 2012.

[72] LAWRENCE, S. R., AND MORTON, T. E. Resource-constrained multi-project scheduling with tardy costs: Comparing myopic, bottleneck, and resource pricing heuristics. European Journal of Operational Research 64, 2 (1993), 168–187.

[73] LENSTRA, J., KAN, A. R., AND BRUCKER, P. Complexity of machine scheduling problems. In Studies in Integer Programming, P. Hammer, E. Johnson, B. Korte, and G. Nemhauser, Eds., vol. 1 of Annals of Discrete Mathematics. Elsevier, 1977, pp. 343 – 362.

[74] LEUNG, J. Handbook of Scheduling: Algorithms, Models, and Performance Analysis. Chapman & Hall/CRC Computer and Information Science Series. CRC Press, 2004.

[75] LIAN, Z., JIAO, B., AND GU, X. A similar particle swarm optimization algorithm for job-shop scheduling to minimize makespan. Applied Mathematics and Computation 183, 2 (2006), 1008 – 1017.

[76] LIEPINS, G. E., HILLIARD, M. R., PALMER, M., AND MORROW, M. Greedy genetics. In Genetic algorithms and their applications: proceedings of the second International Conference on Genetic Algorithms: July 28-31, 1987 at the Massachusetts Institute of Technology, Cambridge, MA (1987), Hillsdale, NJ: L. Erlhaum Associates, 1987.

[77] LIN, T.-L., HORNG, S.-J., KAO, T.-W., CHEN, Y.-H., RUN, R.-S., CHEN, R.-J., LAI, J.-L., AND KUO, I.-H. An efficient job-shop scheduling algorithm based on particle swarm optimization. Expert Systems with Applications 37, 3 (2010), 2629 – 2636.

[78] LÓPEZ-CAMACHO, E., TERASHIMA-MARÍN, H., ROSS, P., AND OCHOA, G. A unified hyper-heuristic framework for solving bin packing problems. Expert Systems with Applications 41, 15 (2014), 6876–6889.

[79] MAHDI, W., MEDJAHED, S. A., AND OUALI, M. Performance analysis of simulated annealing cooling schedules in the context of dense image matching. Computación y Sistemas 21 (09 2017), 493 – 501.

[80] MANNE, A. S. On the job-shop scheduling problem. Operations Research 8, 2 (1960), 219–223.

[81] MARINGER, D. Portfolio Management with Heuristic Optimization (Advances in Computational Management Science). Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2006.

[82] MARTIN, P., AND SHMOYS, D. B. A new approach to computing optimal schedules for the job-shop scheduling problem. In International Conference on Integer Programming and Combinatorial Optimization (1996), Springer, pp. 389–403.

[83] MEI, Y., ZHANG, M., AND NYUGEN, S. Feature selection in evolving job shop dispatching rules with genetic programming. In Proceedings of the Genetic and Evolutionary Computation Conference 2016 (2016), ACM, pp. 365–372.

[84] MENCÍA, C., SIERRA, M. R., AND VARELA, R. Depth-first heuristic search for the job shop scheduling problem. Annals of Operations Research 206, 1 (Jul 2013), 265–296.

[85] MIRSHEKARIAN, S., AND ŠORMAZ, D. N. Correlation of job-shop scheduling problem features with scheduling efficiency. Expert Syst. Appl. 62, C (Nov. 2016), 131–147.

[86] MONTAZERI, M. Hhfs: Hyper-heuristic feature selection. Intelligent Data Analysis 20, 4 (2016), 953–974.

[87] MORTON, T., AND PENTICO, D. W. Heuristic scheduling systems: with applications to production systems and project management, vol. 3. John Wiley & Sons, 1993.

[88] MULUK, A., AKPOLAT, H., AND XU, J. Scheduling problems — an overview. Journal of Systems Science and Systems Engineering 12, 4 (Dec 2003), 481–492.

[89] NOWICKI, E., AND SMUTNICKI, C. A fast taboo search algorithm for the job shop problem. Management Science 42, 6 (1996), 797–813.

[90] ORTNER, M., DESCOMBES, X., AND ZERUBIA, J. An adaptive simulated annealing cooling schedule for object detection in images. Research Report RR-6336, INRIA, 2007.

[91] PANWALKAR, S. S., AND ISKANDER, W. A survey of scheduling rules. Operations research 25, 1 (1977), 45–61.

[92] PENG, B., LÜ, Z., AND CHENG, T. A tabu search/path relinking algorithm to solve the job shop scheduling problem. Computers & Operations Research 53 (2015), 154 – 164.

[93] PILLAY, N., AND QU, R. Hyper-Heuristics: Theory and Applications. Natural Computing Series. Springer International Publishing, 2018.

[94] PINEDO, M. L. Scheduling: Theory, Algorithms, and Systems. Springer, 2016.

[95] PINEDO, M. L. Scheduling: Theory, Algorithms, and Systems. Springer, 2016.

[96] RABELO, L. C., AND JONES, A. Job shop scheduling. Encyclopedia of Operations Research & Management Science (2001), 419 – 429.

[97] RODRÍGUEZ, J. V., PETROVIC, S., AND SALHI, A. A combined meta-heuristic with hyper-heuristic approach to the scheduling of the hybrid flow shop with sequence dependent setup times and uniform machines. In Proceedings of the 3rd Multidisciplinary International Conference on Scheduling: Theory and Applications. MISTA: Paris, France (2007), pp. 506–513.

[98] ROPKE, S., AND PISINGER, D. An adaptive large neighborhood search heuristic for the pickup and delivery problem with time windows. Transportation science 40, 4 (2006), 455–472.

[99] ROSS, P., SCHULENBURG, S., MARÍN-BLÄZQUEZ, J. G., AND HART, E. Hyper-heuristics: learning to combine simple heuristics in bin-packing problems. In Proceedings of the 4th Annual Conference on Genetic and Evolutionary Computation (2002), Morgan Kaufmann Publishers Inc., pp. 942–948.

[100] SABAR, N. R., AND KENDALL, G. Population based monte carlo tree search hyper-heuristic for combinatorial optimization problems. Information Sciences 314 (2015), 225–239.

[101] SATAKE, T., MORIKAWA, K., TAKAHASHI, K., AND NAKAMURA, N. Simulated annealing approach for minimizing the makespan of the general job-shop. International Journal of Production Economics 60-61 (1999), 515 – 522.

[102] SEL, Ç., AND HAMZADAYI, A. A simulated annealing approach based simulation-optimisation to the dynamic job-shop scheduling problem. Pamukkale University Journal of Engineering Sciences 24, 4 (2018).

[103] SEO, M., AND KIM, D. Ant colony optimisation with parameterised search space for the job shop scheduling problem. International Journal of Production Research 48, 4 (2010), 1143–1154.

[104] SIM, K., AND HART, E. Generating single and multiple cooperative heuristics for the one dimensional bin packing problem using a single node genetic programming island model. In Proceedings of the 15th Annual Conference on Genetic and Evolutionary Computation (New York, NY, USA, 2013), GECCO '13, ACM, pp. 1549–1556.

[105] SMITH-MILES, K., AND LOPES, L. Measuring instance difficulty for combinatorial optimization problems. Computers & Operations Research 39, 5 (2012), 875–889.

[106] STORER, R. H., WU, S. D., AND VACCARI, R. New search spaces for sequencing problems with application to job shop scheduling. Management science 38, 10 (1992), 1495–1509.

[107] TAILLARD, E. Benchmarks for basic scheduling problems. European Journal of Operational Research 64, 2 (1993), 278 – 285. Project Management anf Scheduling.

[108] TAY, J. C., AND HO, N. B. Evolving dispatching rules using genetic programming for solving multi-objective flexible job-shop problems. Computers & Industrial Engineering 54, 3 (2008), 453–473.

[109] TSANG, E. Constraint based scheduling: applying constraint programming to scheduling problems. Journal of Scheduling 6, 4 (2003), 413–414.

[110] VAN LAARHOVEN, P. J. M., AARTS, E. H. L., AND LENSTRA, J. K. Job shop scheduling by simulated annealing. Operations Research 40, 1 (1992), 113–125.

[111] WAGNER, H. M. An integer linear-programming model for machine scheduling. Naval Research Logistics Quarterly 6, 2 (1959), 131–140.

[112] WANG, L., CAI, J.-C., AND LI, M. An adaptive multi-population genetic algorithm for job-shop scheduling problem. Advances in Manufacturing 4, 2 (2016), 142–149.

[113] WANG, L., AND ZHENG, D.-Z. An effective hybrid optimization strategy for job-shop scheduling problems. Computers & Operations Research 28, 6 (2001), 585–596.

[114] WANG, Y., ET AL. A new hybrid genetic algorithm for job shop scheduling problem. Computers & Operations Research 39, 10 (2012), 2291–2299.

[115] WEI, L., ZHANG, Z., ZHANG, D., AND LEUNG, S. C. A simulated annealing algorithm for the capacitated vehicle routing problem with two-dimensional loading constraints. European Journal of Operational Research 265, 3 (2018), 843–859.

[116] WENQI, H., AND AIHUA, Y. An improved shifting bottleneck procedure for the job shop scheduling problem. Computers & Operations Research 31, 12 (2004), 2093–2110.

[117] WU, S. D., STORER, R. H., AND PEI-CHANN, C. One-machine rescheduling heuristics with efficiency and stability as criteria. Computers & Operations Research 20, 1 (1993), 1–14.

[118] XIONG, H., FAN, H., JIANG, G., AND LI, G. A simulation-based study of dispatching rules in a dynamic job shop scheduling problem with batch release and extended technical precedence constraints. European Journal of Operational Research 257, 1 (2017), 13–24.

[119] YU, H.-X., WU, A., AND ZHENG, W.-S. Unsupervised person re-identification by deep asymmetric metric. IEEE Transactions on Pattern Analysis and Machine Intelligence (2018).

[120] ZHANG, C., LI, P., GUAN, Z., AND RAO, Y. A tabu search algorithm with a new neighborhood structure for the job shop scheduling problem. Computers & Operations Research 34, 11 (2007), 3229–3242.

[121] ZHAO, F., ZHANG, J., ZHANG, C., AND WANG, J. An improved shuffled complex evolution algorithm with sequence mapping mechanism for job shop scheduling problems. Expert Systems with Applications 42, 8 (2015), 3953 – 3966.

[122] ZHOU, Y., YANG, J.-J., AND ZHENG, L.-Y. Hyper-heuristic coevolution of machine assignment and job sequencing rules for multi-objective dynamic flexible job shop scheduling. IEEE Access 7 (2019), 68–88.

Curriculum Vitae

Fernando Garza Santisteban was born in Monterrey, México, on August 24, 1990. He earned the Chemical Engineering degree from the Instituto Tecnológico y de Estudios Superiores de Monterrey, Monterrey Campus in December 2012. He was accepted in the graduate program in Computer Science in August 2017.

This document was typed in using LaTeX 2_ε[1] by Fernando Garza Santisteban.

[1]The template `MCCi-DCC-Thesis.cls` used to set up this dissertation was prepared by the Research Group with Strategic Focus in Intelligent Systems of the Instituto Tecnológico y de Estudios Superiores de Monterrey, Monterrey Campus.