Instituto Tecnologico y de Estudios Superiores de Monterrey

Monterrey Campus

School of Engineering and Sciences


# Towards the Generation of Heuristics for the Job Scheduling Problem via Crowd Computing: A Video Game Approach

A thesis presented by

**José Martín Mendoza Leal**

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, October, 2021

# Dedication

I would like to dedicate this thesis to my family. Thanks for all your unconditional confidence, support, patience, and encouragement. You were my main motivation when pushing through this work.

# Acknowledgements

I would like to express my deepest gratitude to all those who have been side by side with me. My friends which supported me on many steps of the way. My family that was really helpful to me in many of the stages I went through. My thesis advisor who helped me on the making of this thesis and was also understanding of the problems I encountered. Thanks to Tecnológico de Monterrey for support on tuition and CONACyT with the support for living.

# Towards the Generation of Heuristics for the Job Scheduling Problem via Crowd Computing: A Video Game Approach

by

**José Martín Mendoza Leal**

## Abstract

This thesis was conducted for the Master's in Computer Science Program within the research line of Bio-inspired algorithms with the objective of demonstrating that heuristics for the Job Shop Scheduling Problem (JSSP) can be generated from strategies used by humans when solving the same type of problem, and that this can be done by crowdsourcing using video games. Heuristics are discussed in the computer science field and the psychological field. In both, a heuristic is a quick strategy for decision making when solving a problem which gives up the accuracy of the solution in exchange for obtaining an answer faster. Given that humans tend to use heuristics naturally, an analysis of their behavior can be done to identify these heuristics. The JSSP is an optimization problem within the Scheduling domain where different jobs, composed of activities of different types, must be completed by assigning each of them a position in time in any of the different available machines that are of the same type than the activity, with the intention of finalizing all jobs in the minimum time possible. Scheduling problems including the JSSP are seen in many manufacturing processes and supply chain systems, making them of high interest for companies. Computationally, the JSSP is a hard problem of non-deterministic polynomial time (problems that in order to find the optimal answer, a large amount of computational time is required, making it unfeasible to find a solution when these problems are large) which is why techniques like heuristics are needed to solve big versions of this problem. If people are given the JSSP, in many cases, they will naturally start using heuristics to try to solve it. With this in mind, an analysis designed for identifying heuristics was applied on the behavior of 21 individuals when solving the JSSP.

Crowdsourcing is the use of external people to a project in which the intention is to obtain goods or, in our case, data from a group of participants. In this thesis, the generation of data from humans solving JSSPs was crowdsourced using a video game by making people to solve JSSPs inside the game. For this, different JSS problems were gamified, which means to add typical elements of a game like points and score to an activity, so these could be presented as features in a video game. The gamification of activities has been proven to motivate and increase engagement from the participants, being an important factor that influenced the use of a video game for this thesis. A video game that included the gamified JSSPs was given to play to 21 students. The movements that the players used to solve the JSSPs while playing were collected to be later analyzed. This research shows how video games can be used to gather data from humans when solving JSSPs, and then transforming those solution processes into heuristics by using Machine Learning (ML) algorithms.

Machine Learning algorithms use experience to create models that simulate a behavior and make decisions, which means that analyzing the movements of human players will create methods intended to emulate their behavior. In order to complete this project, first,

experimentation was made on ML algorithms to test their capacity to replicate the behavior of existing heuristics in the literature by training ML models with data generated using those heuristics and under different scenarios. ML algorithms used to create the models were: Decision Trees, Multi-Layer Perceptron, K-neighbors Algorithm, Support Vector Machines and Random Forest Algorithm. After knowing the effectiveness of training and testing these ML algorithms, Decision Trees, Support Vector Machines and Random Forest Algorithms were selected as the better algorithms for continuing the research, and then used to train models with the data collected from the players of the video game. An analysis was conducted on the accuracy on the ML algorithms emulating the players, and aiming at understanding their behavior and strategies for solving problems that can be used as heuristics. The behavior of the ML models were visualized and interesting patterns on how humans solved the JSSP were detected, and promising results were obtained. Some of the heuristics obtained from humans were compared against common heuristics, and interesting conclusions were drawn. This thesis provides the initial steps for generating heuristics from humans when solving difficult computational problems, and leaves and open space for future research to enhance and produce new solution models.

# List of Figures

# List of Tables

# Contents

# Chapter 1

# Introduction

In the computer science field there are problems of "Non-deterministic polynomial time" which can be solved in polynomial time by a non-deterministic Turing machine [30] but no algorithm has been found yet that can solve them in polynomial time on a deterministic computer. The problem with algorithms that are not of polynomial time is that the computational effort needed to solve them grows exponentially relative to the size of the problem. The solutions of many of these problems are very helpful to many other fields of study, but the optimal answers can not always be obtained as problems become large and need a big and unavailable amount of computer power. As there is still the need for a solution to the problems but considering that the optimal answer is unattainable, strategies were created that can obtain "good enough" solutions instead. Strategies of this kind are called heuristics, which consist of simple and quick instructions to choose between available movements when facing a given problem. Heuristics obtain a solution to a problem using fewer operations than the algorithm designed to solve the problem optimally.

The Job Shop Scheduling Problem (JSSP) is a domain of scheduling problems. Scheduling problems consist on allocating activities or processes at a determined place and time considering some restrictions of various types. The most relevant machine environments of the deterministic scheduling form, which are given by Springer [35], include: Flow-shop, Flexible Flow-Shop, Job Shop, Flexible Job and Open-Shop. In this thesis the Job Shop Scheduling Problem will be used to continue the work from Garza et al. [19]. The JSSP consists of completing different jobs composed of activities by assigning each one a time in an specific machine so the activity can be processed; common rules include that a machine can only do one activity at once and that the activities must be done in order, meaning that one must be completed before starting the next one, if these are from the same job. Garza et al. [19] remark that supply-chain based industries and many manufacturing processes use scheduling in their operations in a way that some of their proceedings can be transformed into a JSSP. This means that finding approaches to solve JSSPs will optimize the companies' operation. The improvement of these systems causes reduction in overhead and costs, which makes the refinement of solutions for the JSSP of high interest. Finding an optimal solution for a JSSP has a computational cost of non-deterministic polynomial time (NP-Hard), which means that when problems are large it is not feasible to obtain the optimal answer. Strategies such as heuristics were created for generating approximate answers to the optimal, by giving up finding the best one, in exchange for finding a "good enough" one.

Heuristics are also defined in the psychology field similarly to the computer science field, and hey are described as simple and efficient rules to make a decision when confronted with a problem. Heuristics are usually used by humans when confronted with problems where they lack information or simply when the problem is too complex. Gigerenzer [23] states that heuristics explain the human processes of intuition and habit, and that these heuristics are often unconscious. Solving a JSS problem can be a complex activity when the problem is large, thinking on all possible combinations cannot be done in a short amount of time. This would mean that a common response from a human confronted with a JSSP would be to use heuristics. The quality of their heuristics will depend on the experience and understanding that the human has with the problem, and on the human itself (own experience, creativity, etc..).

To gather people for solving any problem there is usually a need to give something in exchange, which can be money, prestige, entertainment, etc. Games are an activity in which, in many cases, humans must solve problems but can also be a source of entertainment at the same time. There is a technique called gamification which consists of adding game elements to a problem to motivate solvers to work/collaborate with it. This thesis consists of using a video game where JSSPs need to be solved, and people will be asked to play the game in exchange for entertainment. One of the goals of this research was to gamify the JSSPs, and the resulting video game was used to gather data that was eventually analyzed to try to replicate humans' behavior when solving the JSSP in form of a heuristic.

Generating models of how players from a game behave is a study presented by Bakkes [3] called "Player Behavioral Modelling". Studies from this topic have had many uses, including the generation of heuristics from players data. Silva et al. [38] transformed players data into regular expressions by matching sequences of movements to get heuristics for the 2D bin packing problem. Ross et al. [37] used decision trees to store the behavior from human players when helping an evolutionary algorithm which was also solving the 2D bin packing problem. Johns et al. [26] also used a decision tree for finding heuristics for helping an evolutionary algorithm but in this case it was solving water distribution optimization problems. Data to train the decision tree was also obtained from humans doing the same activity. These researchers express that "... most decisions made by an expert user will be based on intuition and 'feel' rather than explicit rules. Therefore, in this paper we introduce the use of machine learning as a mechanism to learn user behavior from interaction and to embed knowledge within the EA.". They explain that Machine Learning (ML) is a technique that teaches computers what comes naturally from humans and learn from experience, and that is why it is a good idea to use them for this kind of problems.

ML is a technique for creating systems that learn automatically through data by identifying patterns in it. Witten [44] explains how many machine learning algorithms are needed since no single one of these is able to identify all possible patterns of interest in data. ML algorithms were used for this project to try to identify the pattern that the humans use while solving JSS problems and replicate it. If the human that the ML algorithm was mimicking used a heuristic when solving the JSSP, then the pattern that the ML algorithm identifies can be labeled as a heuristic. This is how a heuristic would be obtained from human behavior in this thesis. The specific ML technique to use should depend on the data distribution, but the data distribution is unknown as it will depend on the features and the heuristic. ML techniques use different approaches to try to accommodate the data and predict the labels of new points.

As stated by Bonaccorso [4], these methods are usually based on statistical analysis, like using linear regression, hyperplane bisection or divisions by information gain, etc. Technique examples are Neural Networks, K-Neighbors, Support Vectors, Decision Trees, among others. As the labels from the data that will be used for training are already known, ML algorithms will be used with supervised training.

As the objective of this project is to find ways to obtain heuristics that come from humans, the best ML technique to use should be the one that behaves more similarly to humans. Neural Networks are inspired on the brain, but the actual operations and reading of it is far from what humans, at least consciously, do. K-Neighbors uses all the data at the same time to decide, but humans are not able to do this when the amount of data is too large. Support Vectors make a division through the hyperplane, but humans can hardly imagine a space that has more than three dimensions and the hyperplane has $n$ dimensions where $n$ is the number of features. On the other hand, decision trees resolve the problem by making decisions per feature and, depending on the characteristic of the feature it chooses, makes different decisions which can be used to further analyze other features and their effect on the data, or to conclude. This last technique is the one that could better resemble a human's way of thinking.

Even if decision trees could better resemble humans, these would need to have all the features that the human thinks of when solving the JSSP to imitate them completely. There is no way to assure that all possible features that a human can think of were considered in this project. Also, there is no guarantee that all humans made decisions that can be replicated with a decision tree; therefore, many ML techniques were tested.

## 1.1 Problem Statement and Motivation

Heuristics are not capable of always obtaining the optimal answer, so there is always room for improvement, but there is no defined method for getting better heuristics as these come in many shapes and forms. The only way to obtain better heuristics is to create more of them using different methods and compare them to the ones that already exist, then there is a possibility to find a better one. It is important to mention that just generating heuristics at random, or without any focus on solving the problem, naturally reduces the probability of obtaining a good heuristic, and consequently good solutions. This is why getting a method that generates heuristics that are created with the objective of solving a specific problem is of high value. The hypothesis of this thesis is that heuristics can be obtained from human players by analyzing how they solve a problem, in this case the Job Shop Scheduling Problem (JSSP). Humans are a good source of heuristics as we are intelligent beings that already use heuristics naturally. We think that exposing humans to solving JSSPs will likely trigger good solving strategies and heuristics like in many cases.

The reason the JSSP was chosen for this thesis was to continue an internal university research by Garza et al. [19] who solved JSSPs using hyper-heuristics. As heuristics are the main components of hyper-heuristics creating more can be used to get different hyper-heuristics and results which can be better. The JSSP also continues to be a problem of high interest for the industry because there are multiple real-life problems that can be directly transformed into a JSSP. Finally, the JSSP has a good level of complexity for this kind of experiment, as it is hard enough to have many different answers, but not too complex for humans

to get confused when solving it.

JSSPs were gamified for creating the video game that was used to obtain the data. Gamification has been used for many purposes, one of them being collecting data from crowdsourcing (CS). Results from gamifying activities in crowdsourcing are expressed by Morschheuser et al. [32] who analyzed various studies and different gamification implementations. They found out that all of these studies had positive results in the four types of crowdsourcing (crowd-processing, -rating, -solving, and -creating). Studies compared the non-gamified approaches with the gamified ones and the reports showed an increase on engagement, output quality, etc. For this thesis, quality work from people is important as heuristics generated come directly from the strategies the players are using, and the more humans engage with the game and try to get better solutions for the JSSPs the better their strategies become as well as the heuristics we could generate, which shows the importance of gamification in this project. Knowing the importance of the video game, key points proposed by Desurvire et al. [10] were considered; these included:

- Minimize player's fatigue by varying activities and pacing during the game.
- Provide clear goals.
- Game play should have multiple ways to win.
- Player should not be penalized repeatedly for the same failures.
- Pace of the game should apply pressure but not frustrate the player, this for being able to entertain the user and for him to want to give the best answer he can.
- Vary the difficulty so players are able to gain mastery of the game (this part is very important as we want players to use new strategies to solve the problem).
- Game should react in a consistent, challenging and exciting way to the players actions (appropriate music).
- Shorten learning curve by following trends set by the gaming industry.
- Controls should be intuitive but also customizable.
- Provide immediate feedback for user actions.
- The player should be able to turn on and off the game.

These characteristics were evaluated for the game created to analyze its playability.

The data that can be obtained from the video game are the decisions a player took in each state of the game. In order to transform this information into a heuristic, ML algorithms were used as they make decisions based on what they learn from experience. There are many ML algorithms available, and to choose the best one for a case the distribution of the data should be known. As this is not known, many algorithms were tested. There was a prediction that the

data will more likely be better fitted on a decision tree as it makes multiple small decisions as a human usually does, but in order to work properly it must consider all the variables that the human does. As there is no way to ensure that all features that the human brain considers consciously and unconsciously are obtained, ML algorithms that change the perspective on how the features are viewed were used. This way transformations and combinations of features may create the features required. Data from humans was analyzed with decision trees, neural networks, support vector machines, k-neighbors algorithm among other techniques. Visualizing the trained ML algorithms showed how the players were solving the JSSPs.

## 1.2 Hypothesis

A model using video games and crowd-sourcing can be a way towards the generation of heuristics for the JSS problem by using the information from playing records of human players and analyzing it with a variety of Machine Learning methods.

These are the questions to solve:

- How can a video game be used to get information on how humans solve JSSPs?
- What is an effective way to gamify the JSSP in a way that it is engaging to participants?
- How can ML algorithms use data obtained from JSSPs being solved to generate heuristics?
- How can applying ML algorithms to analyze information obtained from the game be used to generate heuristics from humans?
- What kind of strategies do humans use to solve JSSPs? What is the type of data that can be obtained from them? What are the differences between heuristics human use and commonly used heuristics for the JSSP?

## 1.3 Objectives

The general objective of this thesis is to work on the initial steps towards the generation of heuristics for the Job Shop Scheduling Problem (JSSP) using data obtained from crowd-sourcing via video games and analyzed with Machine Learning techniques.

Particular objectives:

- Demonstrate that ML algorithms can replicate behavior from existing heuristics by analyzing data of their movements.
- Show that the JSSP can be represented in a video game which is engaging to human users

- Show that data obtained from a video game in which the JSSP is solved can be used with ML algorithms to replicate human behavior and generate heuristics.
- Provide an analysis on ML algorithms on their capacity to accomplish the objective of the thesis.
- Prove that doing video games and gathering data from players is a feasible way to get heuristics for the Job Shop Scheduling Problem.

## 1.4 Solution Overview

The problem in this thesis is to produce a method that is able to generate heuristics using data obtained from humans solving a problem in a video game. The solution proposed was to use ML algorithms as these are able to define a behavior by learning from experience. The intention consists on obtaining data of what action was chosen on a given state by the player and make the ML algorithm try to choose the same options when given the same states. To do this, the first step was to define the features that compose the data obtained from solving the JSSPs. ML algorithms used were Decision Trees, Multi-Layer Perceptrons, K-neighbors algorithm, Support Vector Machines and Random Forests. Decision Trees were used as these have been used before in similar researches like the one from Johns et al. [26] or the one from Ross et al. [37]. But as Witten [44] expressed, when there is no knowledge of the distribution of the data, there is no way of getting the right answer on which ML algorithm is the better one to use, which is why others were also tested. These ML algorithms were then analyzed with data generated from common heuristics (existing methods in the literature), trying to prove that these can replicate the heuristics' behavior. With this, it was demonstrated that the features that make up the data are enough to detect at least common heuristics, and that the ML algorithms are capable of imitating them.

JSSPs were then gamified and distributed inside a video game focusing on making an easy-to-follow game so players were able to understand the problem they were solving. Many game designs were considered and the one with the best fit to this project was chosen. To gamify the JSS problems, a story was designed and integrated into the environment and placed in their original form, using them to set scores and gain upgrades in the video game.

Data from 21 members an university (students and some faculty) was obtained through the game and then used with the ML algorithms tested at the beginning. Different clusters were created with the data to test multiple options, these clusters extracted the good games from the players with the intention of grouping the games where the same strategy was used. The ML algorithms trained to imitate human behavior were used and declared as the humans' heuristics. Some of these heuristics were visualized for analysis and described in a human understandable way. Also, some heuristics were compared with basic heuristics when solving the JSSP.

## 1.5 Contributions

The main contributions generated in this thesis are the following:

- The main content of the thesis is a basis on JSSP heuristic gathering through data from humans playing video games. Being able to exploit data from video games has a great advantage as these are a common activity which produce large amounts of data. Also, many video games make players solve difficult scenarios as part of the game. Because of this, this thesis could work as a step forward to develop new projects to grow and take advantage of one of the biggest sources of data coming from problems being solved by intelligent beings.
- For completing the current project, a video game platform was created which had the purpose of collecting data of human players solving JSSPs. This platform helps to establish a method for others doing the same or something similar and it can also be used by the community to obtain their data. Also, other video game design options were proposed.
- ML methods were analyzed on their capacity of imitating heuristics for the JSSP and a way of analyzing data from humans to obtain their heuristics based on their strategy when solving the JSSP was created. This is a new concept and it will be relevant for the generation of new methods with the same objective.
- Some heuristics that humans use when solving the JSSP were recognized, analyzed and compared.

## 1.6 Thesis Organization

This paper is organized as follows: Chapter 2: “Background” shows information on the main topics of this thesis and how these have been used in relevant ways to this thesis , Chapter 3: “Solution Model” explains the pieces which form the project and the relationship between them, Chapter 4: “Methodology“ explains the steps that where followed in order to do this research, Chapter 5: “Creating and Testing the Basic Model” talks about the first experiment that was done, Chapter 6: “Making the Video Game” gives the steps followed when making the video game for obtaining the JSSP data and also includes an analysis on how games for problems should be made, Chapter 7: “Process for Generating Heuristics” give the results obtained after analyzing the data with the ML algorithms, which includes the capacity of the ML algorithms to imitate human behavior and building the heuristics, and also the comparison between human heuristics and common heuristics, Chapter 8: Conclusions gives information on what was learned on this project and what can be done in the future.

# Chapter 2

# Background

This chapter includes information about important topics for the development of this thesis. This information gives an insight into what has been done before that could help on testing the hypothesis. These topics include: Heuristics, Job Shop Scheduling Problem, Video Games Characteristics, Crowd-sourcing, Player Behavioral Modelling, Machine Learning and other related works.

## 2.1 Heuristics

Heuristics are inspired on a concept in the field of psychology which describes a theory on how humans think. Knowing more about heuristics in psychology was necessary as heuristics are going to be obtained from humans.

### 2.1.1 Heuristics in Psychology

Gilovich [24] wrote: ”This work, like the heuristics and biases program, stresses the fact that much of the mental life is not the product of deliberate processing, but of quicker, more reflexive processes that are less available to conscious intervention.” In this text Gilovich expresses one of the main concepts that inspired the approach of this thesis, which is that most processes that human make are ”quicker”, ”reflexive” and ”less available to the conscious”. The words quicker and reflexive are the ones that better describe heuristics, and these should be the heuristics that will be looked for. As it is a reflexive process, complex heuristics would not be expected, which is why basic heuristics will be used as a basis. The ”less available to the conscious” phrase is really important to give a reason to why not ask the humans for their strategies they used directly. This is because the might not be aware of the actual strategy that their brain is using.

After understanding a little more about heuristics and how these are in the human mind, the next step is to understand what to expect from heuristics. Gingerenzer [23] explains how: ”heuristics are evaluated against divine ideals, which makes them appear to be all-too-human failures. They refer to three ideals: omniscience, optimization, and universality. Omniscience is the ideal of full knowledge, which is often assumed in theories of human rationality; its modest sister is the ideal that more information is always better, or cannot hurt. Optimization

is the ideal that a best solution for each problem exists and that we know how to find it. Universality is the ideal that this best strategy, such as maximizing expected utility, is universally the same for all problems. Heuristics run counter to these ideals, in that they assume limited knowledge rather than omniscience, their goal is to find a good solution without the fiction of an optimal one, there is no universal heuristic, but an adaptive toolbox with many building blocks from which new heuristics can be constructed." This text shows the main limits of heuristics, and in computer science these limits are similar. This text is a complete definition of why and how are heuristics used (why? because limits, how? by expecting only good solutions and with a toolbox), the "adaptive toolbox" description on the psychology field can describe how humans use a different heuristic for different situations, but, How about different situations in the same problem? Because of this the use of different heuristics on the same problem should be expected from humans. Gingerenzer [23] also explains that heuristics are constructed by: a priority rule, which is the order (gain or probability of gain); a stopping rule, which is when the program will stop searching for options; decision rule, which is to choose the gamble with the more attractive gain.

### 2.1.2 Heuristics in Computer Science

As in all fields, in computer science there is a discussion of what exactly can be called a heuristic. If it is too complicated can it still be defined as a heuristic?, Who sets the limits of when something becomes too complicated? Gere [22] described heuristics as rules of thumb when he used them for solving the JSSP. This simple definition will be the one taken for this thesis. A rule of thumb as a decision made based in experience.

In this thesis, heuristics were created through data. Edelkamp and Lomuscio [12] worked on pattern databases and heuristics. They defined pattern databases as dictionaries for heuristic estimates that store state-to-goal distances in state space abstractions. They explain that this process' effectiveness is dependent on the manual selection of patterns. This process is made to create an automatic model for the creation of heuristics, which means that is similar to what was done in this thesis.

They also proposed to use a genetic algorithm for the training part of their program in order to select heuristics on an abstract space, transforming patterns of the best cases into a function. This work is important as it expresses the features that they needed in order to achieve their goals, which could have been used in this thesis.

## 2.2 Job Shop Scheduling Problem (JSSP)

In this section the JSSP is explained and after many researches where some approaches with heuristics or elements similar to heuristics were used to solve the JSSP, are cited to see how the JSSP is being solved.

To explain the job-shop scheduling problem we first have to formulate it as follows: we have $n$ jobs $(J1, J2...Jn)$ and $m$ different machines $(M1, M2...Mm)$. Each job has a different number $c$ of activities $(A1, A2...Ac)$ which have to be processed sequentially on that order. Each activity has a scheduling time of $k$, which is the time the activity takes to be completed.$Ai$ must have been completely processed before $Aj$ if $i < j$. Activity $Ai$ can be

processed by only one machine, and has to be completed from start to finish for $k$ time once assigned. So we can see that a machine can only process 1 activity at a point in time, and also one activity can only be processed by one machine at a time. Activities and Machines have a type $t$, an Activity can only be processed on Machines with the same type $t$. With this restrictions in mind, the objective of the JSSP problem is to find the order and machines in which all activities $A$ have to be done in order to finish all of the jobs in the minimum possible time.

Other variations for the JSSP which are machine environments of the deterministic scheduling form, are given by Springer [35]. These include:

1. Flow-shop: In this version, each job has the same sequence of processing on the machines where the $i$th activity of a Job most be processed by the $i$th machine.

2. Flexible Flow-Shop: In this variation there are stages in which a number o identical machines (machines of the same type) become available for activities to be done. Each job has its own sequence of processing on stages and each machine can only process one activity.

3. Job Shop: Each Job has its own sequence of processing in machines.

4. Flexible Job Shop: Similar to the Flexible Flow-Shop with its stages, but there is a predefined sequence to be followed.

5. Open-Shop: Each activity from a job has to be processed on a machine in any order.

The JSSP has been tested with heuristics and hyper-heuristics with positive results as shown by Garza, et al. [19] and [20]. They also tested new heuristics comparing them to the best solution found by the following common heuristics (the best solution of a group of common heuristics is called an oracle):

- 1) Put the next available activity into the first available time.

- 2) Most Loaded Machine (MLMACH): Get the machine with the maximum total processing time and place there the activity that will have the least possible scheduling time when placed there. If there is not an activity available for that machine try another machine.

- 3) Maximum Remaining Time (MRTIME): Select the job that takes the most time and get the first activity from it, assign this activity to a machine and continue the same process.

- 4) Earliest Start Time (EST): Find the job that could start first in the current state of the problem, then start the first activity of said job.

Another research done also by Garza et al. [21] tested the use of simulated annealing for creating the hyper-heuristics that they constructed.

A branch and bound algorithm has also been proposed by Brucker et al. [7], where after experimenting with the 10X10 JSS problem they found out that for this a heuristic based

on a priority dispatching rule got the best results. Results where compared against some benchmarks and only obtained good CPU time results on JSSPs with 10 machines or less. This heuristic is applied as follows:

- First get all the operations that are able to be scheduled next (meaning that their predecessors have already been done), put them on a set and name it $C$.
- Select the operation with the minimum calculation for (node value $R$ + processing time) for every type of machine.
- Calculate for each of the operations selected the lower bound for the makespan of the schedule if it is chosen, and choose the one with the minimum lower bound.
- After adding the new item, update your set $C$ with the operations that can be scheduled next (the successors of the operation you added).

Naikunth, et al. [1], proposed computational grids and using what they call ”Nature’s Heuristics” for solving the Job Shop Scheduling Problem. They tried to combine the heuristics named Genetic Algorithm (GA), Simulating Annealing (SA) and Tabu Search (TS) for JSSP. Their research showed that compared to pure GA, GA-SA should have better convergence, and GA-TS should have better search efficiency. They also explained how at least 8 other related researches have shown that hybrid heuristics do have a better performance.

Another perspective on heuristics for the JSSP is presented by Colorni et al. [8] where they presented their own original heuristic called ”ant system”. As it names implies, this heuristic is based on the way ants work. This heuristic uses different agents called ”ants”, and the effectiveness of the algorithm depends on the cooperation of these ants, which they do by the periodical modification of a global trail matrix.

For the JSSP, multiple ants are sent to follow a greedy heuristic, and depending on the individual performace of every ant (if these are having positive or negative results) the path they are following will receive more trail which will mean that more ants will follow that ant. The reason we want more ants following a trail is that each ant will search within the trail’s different paths, that way we are able to try different methods.

Another recent research which includes the comparison of various algorithms for solving the JSSP was done by Syarif et al. [40] where a Genetic Algorithm (GA) they developed was compared to Particle Swarm Optimization (PSO), Upper-level algorithm (UPLA), Differential-based Harmony Search (DHS), Grey Wolf Optimization (GWO), Ant Colony Optimization (ACO), Bacterial Foraging Optimization (BFO), Parallel Bat Optimization (PBA), and Tabu Search (TS). ”The experimental results of the 28 benchmark test problems validated that the algorithms, except ACO, can provide the optimal solution of JSSP. PBA delivers the most impressive performance that solves 26 cases optimally, with the average error equal to 0.05%.” [40]

## 2.3 Video Games

It was important to establish the characteristics of the video game, as we want to make a playable and entertaining game from different JSSP problems. This means we need to know

what a game is in order to convert problems into mini-games. Also, in order to be successful in this method of crowd-sourcing we need to define the characteristics that will make it distributable.

Fabricatore, Rosas and Nussbaum [15] give their ”design prescription” for making a game ”playable”. Their main points are shown below:

- Goals must always be understandable and unambiguous, and should not be too repetitive to avoid monotony and sustain motivation.
- The relationships and dependencies between stages of alternative branches of the game, and between different stages of the same branch must always be clearly understandable by the player.
- Whenever non-linear developments are possible, it must be possible to backtrack after a decision has been made, especially if the decision is wrong and leads to negative consequences.
- Finding alternative branches in a non-linear hierarchy of goals should not be excessively time consuming.
- Linearity is a better option in contexts in which non-linearity could make players go through visited places over and over again, trying to figure out what must be done.
- Minimum information regarding progress should include data about failures, in order to allow the player to learn from his/her own errors.
- Temporarily freezing the game-flow while the player is reading a map may benefit beginners, but compromises the realism of the game.
- Providing help during customization processes regarding the gaming world may reduce game-flow disruptions.

Fabricatore [14] expresses three activities that a game needs to follow in order to involve a player with the game:

- The first one would be to let the player explore the mechanics of the game by himself.
- Next will be to make the player understand when he has to use the different mechanics of the game given a situation.
- The final step for involving the player in the game would be to change common mechanics and affect the player depending on new situations the game proposes.

Federoff [16] gave some important heuristics for evaluating usability in games:

- Game and play mechanics get the player involved quickly and easily.
- Use visual and audio effects to arouse interest.

- Controls should be intuitive and mapped in a natural way.
- Include a lot of interactive props for the player to interact with.
- Design for multiple paths through the game.

Gamification is an strategy that is still being used, as said in the paper from Romanov [36]: “At the same time, gamification has been identified as a viable teaching and learning method in higher education, which might be also appealing to non-professionals and might bridge their knowledge gap in DH.”

## 2.4 Crowdsourcing

Crowdsourcing as described by Brabham [5] is ”...an online, distributed problem-solving and production model that leverages the collective intelligence of online communities to serve specific organizational goals. Online communities, also called crowds, are given the opportunity to respond to crowdsourcing activities promoted by the organization and they are motivated to respond for a variety of reasons”. In this thesis the activity was to play a video game in which the JSSP is being solved, and the motivation would be to get entertained. Strategies for distribution, and the effects of using a game for crowdsourcing were be analyzed in this section.

Juho et al. [32] explained the success of gamification used in crowd-sourcing. They explored multiple cases where gamification was used. Their statistics included information such as how much of the gamification implementation used points and leader-board on their game, and how this improved competition and quality of players. They also showed how all the studies of impact of gamification on crowd-sourcing had positive results. These results come from increase of different features like engagement of participants, quality of the output, reduction of ”cheating”, as well as users of the game denoting that they preferred the gamified style than the non-gamified of what they where using.

Vannella et al. [43] explain the characteristics that a game needs in order to be successful in a crowd-sourcing environment: First, making game-play natural and with familiar video game mechanics. This first step also included using common actions to play the game, like collecting items, mini puzzles, among others, rather than extrinsic tasks.
The second objective is to make the game playable by single player, and that you add some reinforcement for playing the game correctly.
The third one is that the game design should be general in order that any change needed because of linguistic phenomena can be accomplished by simply changing game data.

## 2.5 Machine Learning

Bonaccorso [4] describes the main goal of machine learning as to study, engineer and improve mathematical models that can be trained with context related data to infer the future and make decisions. Another description he gives is ”In other words, an agent (which is a software entity that receives information from an environment, picks the best action to reach a specific

goal, and observes the results of it) adopts a statistical learning approach, trying to determine the right probability distributions and use them to compute the action (value or decision) that is most likely to be successful (with the least error)."

Ayodele [2] states that: "Machine learning is about designing algorithms that allow a computer to learn. Learning is not necessarily involves consciousness but learning is a matter of finding statistical regularities or other patterns in the data. Thus, many machine learning algorithms will barely resemble how human might approach a learning task. However, learning algorithms can give insight into the relative difficulty of learning in different environments."

Both authors express how machine learning is a predictor that learns from experience as it is needed for this thesis, but they also state that ML algorithms don't work as humans, but as statistics analyzers. It is true that answers that were obtained in this project might not resemble exactly how humans think, in reality the only classifier that has a possibility to do that is the binary tree. The reason this project still used machine learning is because even if heuristics obtained from humans are not readable it is important to know that a pattern does exist, so in future projects these can grab on to this fact and continue to research using better methods.

"Feature Engineering Is The Key At the end of the day, some machine learning projects succeed and some fail. What makes the difference? Easily the most important factor is the features used. Learning is easy if you have many independent features that each correlate well with the class. On the other hand, if the class is a very complex function of the features, you may not be able to learn it." this was written by Domingos [11], and its really important to consider that the definition of the features its of high impact on the development of this thesis.

A problem in mind is to know which specific ML algorithm to use in this scenario where the type of data is unknown, Witten [44] states that"In the infinite variety of possible datasets there are many different kinds of structures that can occur, and a data mining tool—no matter how capable—that is looking for one class of structure may completely miss regularities of a different kind, regardless of how rudimentary those may be. The result is a baroque and opaque classification structure of one kind instead of a simple, elegant, immediately comprehensible structure of another". This means that many ML models should be tested to see how each one adapts to the data and serach for the one that "fits".

## 2.6 Player Behavioral Modelling for Video Games

"Player behavioural modelling is a research area in game playing that is gaining attention from both game researchers and game developers. It concerns generating models of player behaviour and exploiting the models in actual play" (Bakkes et al.) [3].

Player behavioral modelling has been used for many things, including the improvement of AI in video game as in researches from Pfau et al. [33] [34]; the recollection of solutions to problems as done by Mavandadi et al. [31] who collected labels from players to blood cells images, or the study from Estrada et al. [13] where information for decision making in refugee aid deployment was made by learning from players making decisions on a simulation.

Analysis in video games has also been used to obtain heuristics for solving optimization problems as what is being done for this thesis, it can be seen in studies from Silva et al. [38] and Ross et al. [37] where heuristics for solving the 2D bin packing problem are looked for.

Silva et al. [38] used data from players to detect patterns and build regular expressions by matching sequences that were established with the movements from players in a mini game that was created by Silva. Ross et al. [37] used data from players to find heuristics for an evolutionary algorithm which is made to solve the bin packing problem, a decision tree, that uses as an input simple features, was trained with human decisions over the problem to substitute them with the decision tree when solving new problems.

Another paper that uses decisions trees is the one from Johns et al. [26]. These are used similarly, which is to substitute decision making that humans commonly make over results from an evolutionary algorithm to guide the evolutionary process, with decision making from a decision tree. In this case the problem that the evolutionary algorithm was solving was the optimization of a design for a water distribution where there are people that guide an evolutionary algorithm to get to a better solution, solving minor conflicts in the solutions proposed by the algorithm. Johns does mention that using ML algorithms for this types of problems is the logical step as these have the ability to replicate complex decision making.

Player modelling has also been used to predict outside game behavior of the user, as in the study from Loria et al. [28] who trained a random forest model using players behavioral data to know if that player was about to stop playing the game. It can be seen that ML algorithms based in decision trees are widely selected for behavior interpretation of players.

## 2.7 Related work

In this section, work similar to what is being done in this thesis is discussed. It is important to note that some researchers define hyper-heuristics as a creation of heuristics, and many methods for this have been created using multiple machine learning techniques, but the main difference between the creation of hyper-heuristics and the method proposed is that the main focus in this thesis is the description of the problem using features and the identification of the features that affect the most the decision and not the identification of heuristics in data. Even if features detected are then used to detect heuristics, the difference is that in this project unknown heuristics are also being looked for. The method proposed is able to detect hyper heuristics, but it is also designed to be able to detect unknown heuristics.

This research is a continuation from the research from Garza et al. [19] who made an analysis on using Hyper-Heuristics for solving the JSSP. Hyper-heuristics is an strategy where a different heuristic is used depending on the state of the JSSP being solved. A variety of heuristics means more possibilities of hyper-heuristics, which may become better than the hyper-heuristics already obtained.

The research “Discovering Heuristics And Metaheuristics For Job Shop Scheduling From Scratch Via Deep Reinforcement Learning” from Ekeris et al. [42] has many similarities with what is being done in this thesis, as they also obtain heuristics for the JSSP using data from it. In this research Deep Networks were trained by making them solve JSSPs and feedback is received punishing larger makespan. Some of the features used in this research were also obtained based on what the basic heuristics needed. The main difference is that this thesis did not used players to obtain its data, but it did experimented with its deep network to see if it could replicate basic heuristics.

The way Ekeris et al. [42] replicated heuristics using Deep Networks was by training

them with only the features specific to the heuristic, and in most cases the features were from two different heuristics i.e. longest job first and shortest job first, and the network inclined to replicate the one with the lowest makespan as that is what it was trained to do. In the current thesis, the method used should be able to replicate the heuristics without changing its architecture, which is why features cannot be removed as it was done in this other research.

Good et al. [25] made a video game for getting data from players. This is similar to this project as a game that will be crowd-sourced was also developed with the purpose of collecting information. Their program is called ”Foldit” and they used common missions of popular games to try to gather people to solve problems of protein folding with similar missions. This is a very important note as we want players to enjoy the game we make. In order for them to actually try to solve the problems, we need to make our game similar to something they already like.

Foldit is a visual puzzle game in which the player has a sequence of a protein that is partially folded. The challenge is to find its lowest-energy three dimensional structure. To do this, players can pull, twist, tug the protein as they see how their movements affect the results.

Eurisko by Lenat [27] is a program with many similarities with what we want to do with the data. This program gets heuristics by creating short-cuts through the code by analyzing how the program used functions to get an answer and detecting how many variables change in the process. It also uses a strategy of playing with ANDs and ORs in IFs in order to know if there is a shortcut to get the same answer.

Another topic of interest is the automatic creation of heuristics for games. The paper from Mesentier [9] talks about the generation of ”heuristics for novice players” in games, but does it from data generated by a computer. This is not the same as what is being done is this thesis as in this thesis heuristics are obtained from humans, but it is connected to what is being done here. This paper was important as it showed a method that generated heuristics from game data.

## 2.8 Summary

In this chapter, information about heuristics was learned, how these might be an unconscious or intuitive human behavior, and how these are simple. Many approaches on how the JSSP is solved were described. Information about video games and crowdsourcing was obtained. This information gave precise items to evaluate on this project. Characteristics from the game developed were considered and evaluated.

The information above covered the idea that the game must give the players a chance to practice slowly increasing the difficulty. Because of this, an experiment should be done where the first tries of the players are not analyzed as a heuristic, as players might have not yet developed their strategy.

Machine Learning was also covered. The most important element to consider about ML algorithms are the features defined at the beginning as it was mentioned that features are the most important element for ML. Features tried to cover as much information as possible in order to be able to describe as many heuristics as possible. Many ML algorithms have to be tested to find the right one for this type of problem. Studies where decision trees have been used to generate heuristics based on human players behavior were showed, and these were

taken for validation of what was done in this thesis.

# Chapter 3

# Solution Model

The pieces that conform this project are the following: Job Shop Scheduling Problems (JSSP), Humans, a Video Game, and a Data Analyzer. The relationship between them is shown on the diagram in Figure 3.1, which explains how JSSPs were used on the Video Game on a gamified way (solving JSSPs gave players points to advance within the game), and how Humans were the ones solving these JSSPs when playing the Video Game. Data of the players' movements was gathered by the Video Game and this was sent to the Data Analyzer so it could generate heuristics by trying to imitate Humans' behavior with machine learning algorithms.

## 3.1 JSSP

Job Shop Scheduling, which is the problem explained before about jobs which needed to be completed on machines, was used as the challenge for the Humans playing the Video Game. The JSSPs were gamified by converting the description of the machines, jobs and activities, into a simple graphical representation as shown in Figure 3.2. The player was able to move activities into a machine as long as the JSSP rules were followed. Gamification also included adding a score mechanism based on how many activities were placed correctly so players were motivated to solve the problems as best as they can. These problems were generated with Taillard‘s method [41] using some of the examples that were used on the paper cited, but also generating more simple examples to adapt to the players necessity.

## 3.2 Humans

Humans are the intelligence from which the heuristics created were based on. Their purpose was to play the Video Game and solve the JSSPs. Their data on how they solve the JSSPs was analyzed to obtain the heuristics based on their strategy. The expectation was for them to try their best when playing the game. This is why the game was presented as a challenge with points and inner game upgrades to motivate them.

The plan was to get support from various students in the campus in order to make the tests.

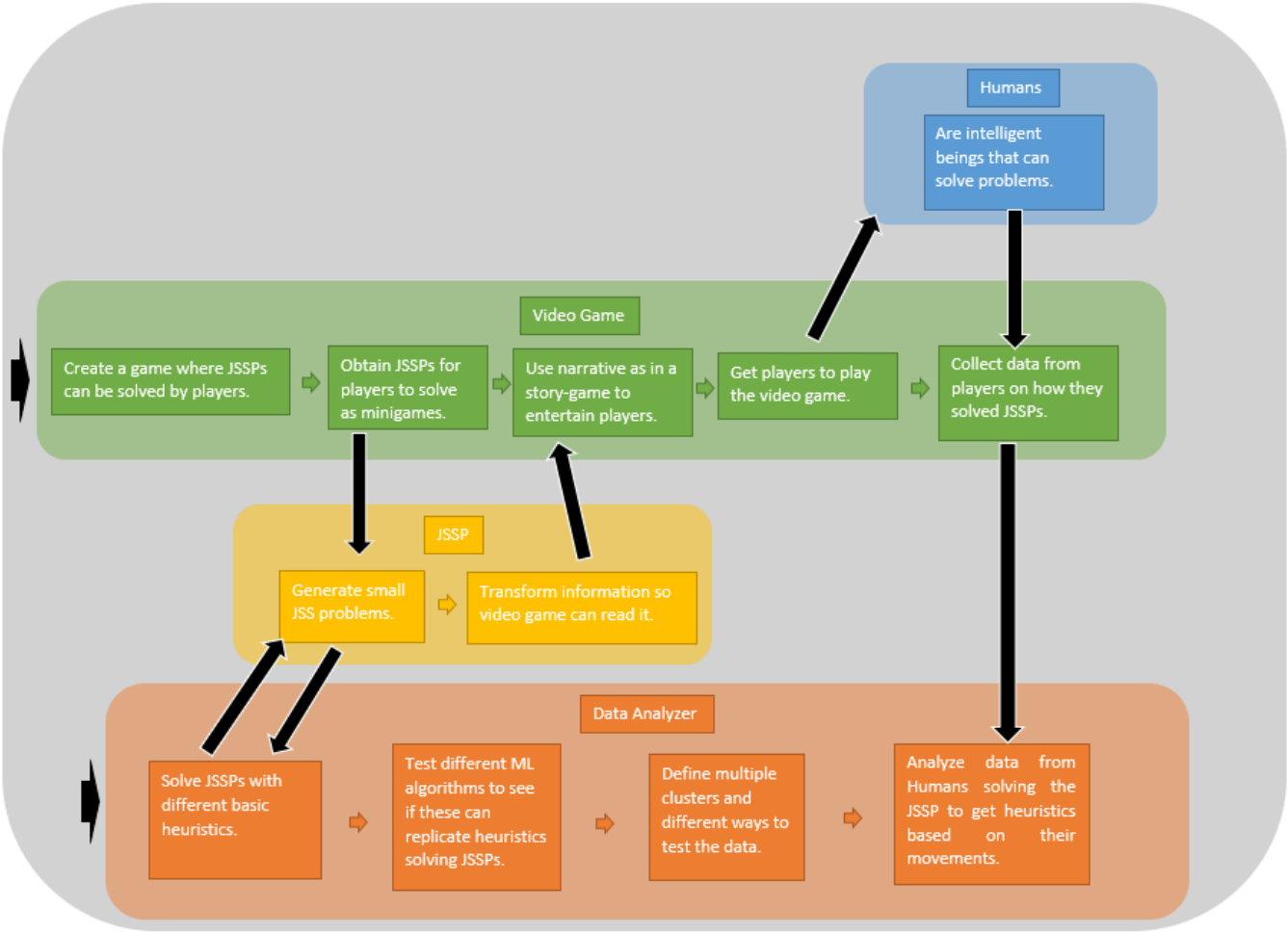


Figure 3.1: Solution Model.

## 3.3 Video Game

This video game was made with the purpose of getting Humans to solve JSSPs willingly in exchange for entertainment. Entertainment was added by creating a virtual world which players oversaw, and as the players solved JSSPs they obtained points and were able to upgrade their world unlocking stories where they were able to see the changes they created. The Video Game was also made highly accessible by making it usable in most browsers (coding it using HTML, JS and CSS) as well as simplifying the distribution, with the purpose of generating commodity between the people that were asked to play it. The game collected data from each movement of an activity that players made and saved it in a way that it was simple to identify the player, the JSSP that was being solved and the order in which each movement was done. Each movement included the activity selected and to which machine and which position it was moved to.

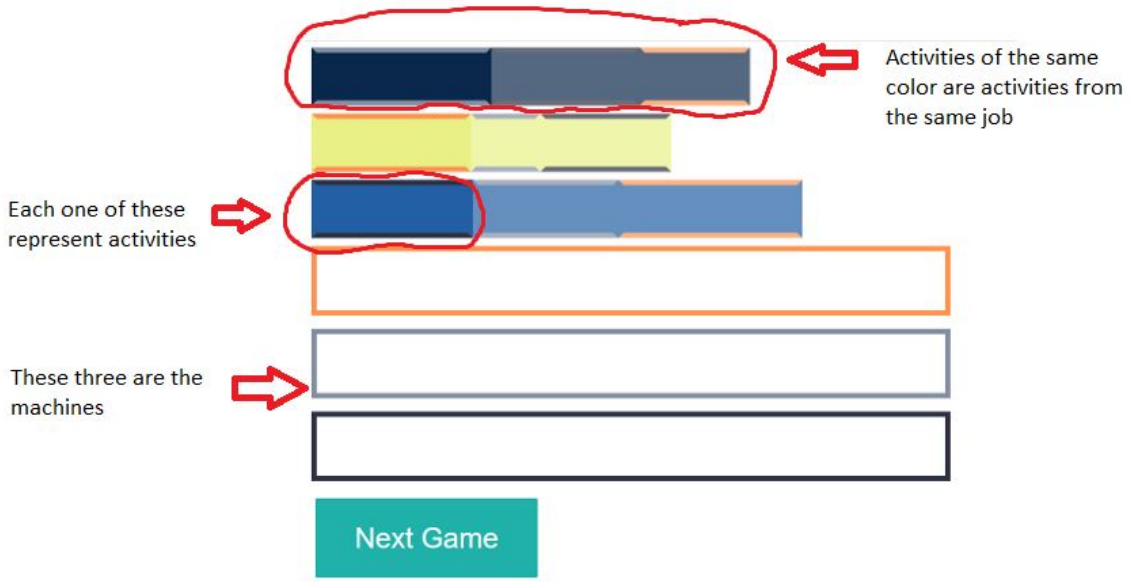


Figure 3.2: Graphical JSSP.

## 3.4 Data Analyzer

Using the data obtained from the game, different Machine Learning algorithms were used to try to find a model which imitates the strategy that humans used to solve the JSSP in the game. The data contained information to replicate humans' movements in the JSSP which was used to obtain two new sets of data. The first set was a "before movement" set which trained the ML algorithms to try and replicate a human's behavior using information of the activities available in a state of a JSSP and information of the current state. The second state was called "after movement" which trained the ML algorithms to imitate the same behavior and also used information of the activities available in a state but with information of what the state of the JSSP would be after each activity would be selected. Results of using both data models were then analyzed. These results contain the heuristics that were planned to be obtained from humans.

## 3.5 Summary

This thesis consists on two main components, a Video Game and a Data Analyzer. The Video Game was made as a way of connecting Humans with the JSSP, as this thesis requires data on how humans solve the JSSP. The Video Game was made by gamifying many JSS problems and adding a narrative so humans can be entertained while playing the game and solving the problems. The second main component is a Data Analyzer which is a component for analyzing the data that was obtained from humans. To create the Data Analyzer different ML

algorithms were tested and also different ways of analyzing the data were defined. At the end, some heuristics were obtained by analyzing with the Data Analyzer, data of Humans solving the JSSP which was collected by the Video Game.

# Chapter 4

# Methodology

This chapter explains the process that was followed for the development of this thesis in order to achieve its objectives. Sections include: "Analyzing the problem" which talks about the literature review and the logical process behind some decisions, "basic model" which was about the first experiments that were done to test some machine learning algorithms for making them imitate heuristics, "Developing the video game" where it explains the process on how the game was made, "Generating Heuristics" explains how the data obtained from the video game was used and "Paper writing" which was a step that was done in parallel with the others and elaborates on how this thesis was written.

1. **Analyzing the problem**

   As for every scientific problem, analysis and research was done for this one. The first step consisted of analyzing each of the components of the problem which had a similar description as the hypothesis. The main idea was to obtain heuristics for the JSSP using human information obtained from video games. The first concept to analyze was heuristics, which included what are the main characteristics of the heuristics, and the answer to the questions: Do humans use heuristics? And how do humans use heuristics? After knowing that humans do use heuristics and their characteristics the next important question was to know if it was relevant to obtain heuristics from human behavior, as most heuristics that are already described came from humans. Because of this it was important to know that some heuristics are unconscious and that humans may sometimes not know the heuristic that their mind is using.

   Using a video game to get data is an innovative idea to test out, as the development of this concept can be very beneficial to the science field considering that it can evolve to be useful for getting large amounts of data. This problem was analyzed using the JSSP as this problem seemed to have fitting characteristics for trying it with the objective, such as being difficult enough to have many ways of solving it or being relevant in the computer science field. It was also used to continue the work from Garza et al. [19] who tested the use of hyper-heuristics with the JSSP, so obtaining new heuristics can get to different and better results to that research.

   After understanding the reasons for this project and knowing that there is a possibility of getting heuristics from humans, the next step was to plan how this project was going to be done. As the project consisted of analyzing data of movements from human players

and trying to obtain an agent that replicates their behavior (their heuristic), the better fit in the computer science field were the machine learning (ML) algorithms. Other strategies were considered, for example defining a form for heuristics, which should be an object with values where every heuristic could be represented with the changing of these values, and with the movements of the players to the JSSP, adapt the values of the object so both make the same decisions. The problem with this method would be that it would be difficult to define an object that can represent all heuristics, but also this won't assure that not known heuristics could also be represented. Thus, ML was the final decision. The next thing that was considered was the definition of the features that composed the data. It was decided for the data composition to test using features that could directly identify simple heuristics, for example for the heuristic shortest job first, there should be a feature which is able to identify uniquely the shortest job. Features that identify the state were also added, these were obtained from the thesis of Garza et al. [19].

2. **Creating and Testing the Basic Model (Chapter 5)**

   After it was defined that Machine Learning (ML) algorithms were going to be used and how the features were going to be identified, the next step consisted of defining these features. The general features were already defined, but for the specific ones, simple heuristics had to be specified. Heuristics from the thesis of Garza et al. [19] were the ones used, and from them other features were determined.

   With the features defined, the next thing to consider between using the data "before a movement" or "after a movement". Heuristics usually work before a movement; an example is the heuristic of selecting the smallest activity first where for knowing which is the smallest activity, there is no need to know the effect of selecting said activity. On the other hand, ML algorithms are commonly used analyzing data after the movement, as all features are considered and when analyzing different states it is convenient that most of the feature values are different. As this investigation was trying to get heuristics but using ML it was decided to use both.

   ML algorithms that were going to be used also had to be defined. Many common ones were tested as there was no initial idea of how any would work. ML methods tested are: Decision Trees, Multilayer Perceptrons, K-neighbors classifier, Support Vector Machine and Random Forest. As features were obtained from heuristics, it was expected from the ML algorithms to have high accuracy.

   JSSPs were created using Thaillard's method [41], these were solved by using the heuristics defined, and data with the determined features was generated during that process. ML algorithms analyzed the data and good results were obtained as expected in some of them, which after selecting some of the ML algorithms, gave the green light to continue with the planned strategy.

3. **Developing the Video Game (Chapter 6)**

   The first step in developing the video game was to design it. The main characteristics like usability, availability and difficulty were defined considering the use that this game

will have. After this, different methods of representing the JSSP in a game were analyzed, and it was decided that a simple representation was the best one for the current project.

In the process, a simple version of the game was created to test and get feedback from a few people. This was important feedback to analyze as people feeling comfortable and engaged with the game was a main point in this thesis.

The final version was then created adding some visual cues and improving the design considering comments from the first version. Also, the game was converted to a story game, this means that we added a story and a shop where upgrades that affect the story could be bought with points obtained from solving JSSPs. This game was used to collect data from players solving the JSSP which was then analyzed for obtaining the heuristics.

4. **Process for Generating Heuristics (Chapter 7)**

   ML algorithms tested in Chapter 5 were used to analyzed data generated in the video game described in Chapter 6. In chapter 5, "pre-movement" and "post movement" data were analyzed; but here, there was a need to create ML classifiers for getting positions from the machine because players could assign activities in any position of the machine when heuristics only placed them at the end. ML algorithms showed a moderate accuracy but were able to recognize strategies that humans were probably using.

   Multiple clusters were done to the data to test the ML algorithms in many forms and to obtain new results. Heuristics obtained were visualized, analyzed and compared against the basic heuristics on their capacity to give good solutions for the JSSP. With this it was concluded that ML algorithms are able to obtain heuristics from humans, but that the heuristics acquired were not able to outperform simple heuristics.

5. **Thesis Writing**

   This thesis was written alongside the development of the experiments. It was a complicating step as it not only had to include everything that was done, but also openings to what future projects could be. As this is the first time analyzing this approach, it was important to show possible paths to the readers to have multiple options if the current path is not good enough.

## 4.1 Summary

The thesis was conducted first by doing research on the concepts and defining how each of the objectives were going to be accomplished. The objective is to get heuristics for the Job Shop Scheduling Problem (JSSP) using data obtained from crowd-sourcing via video games. After recognizing the viability of the objective, an approach for how heuristics were going to be obtained from data was defined which was to use ML algorithms. The next step of the thesis consisted on testing ML algorithms to see if these were able to replicate basic heuristics. After obtaining a positive answer, this approach was approved. The next step was to develop the

video game to obtain the data. In this step, many game designs that could accomplish this objective were made, and a story game which uses a graphical version of the JSSP for players to solve was developed. The video game was distributed among some students on the campus and data of how they solved the JSSPs was collected. This data was then analyzed with the ML algorithms to obtain a simulation model for the players' behavior which was analyzed to recognize heuristics used by players.

# Chapter 5

# Creating and Testing the Basic Model

The objective of this thesis is to obtain heuristics for the JSSP using data obtained from crowd-sourcing via video games and then analyzed with Machine Learning (ML) techniques. This chapter focuses on analyzing the method proposed to accomplish this objective and its ability to obtain ML models that imitate common existing heuristics and some hyper-heuristics.

This method was evaluated with its capacity of creating ML models that can be used to replicate the behavior of single common existing heuristics and hyper-heuristics. The method consists on creating models capable of identifying if a single activity of the JSSP on a specific state would have been chosen by the heuristic or hyper-heuristic to replicate. To do this, ML algorithms needed to recognize the characteristics or features of an activity that were relevant to the decision making of the heuristic.

In order to generate the ML models, these were trained using information of single activities in different states of the JSSP. Each row of data represented an activity, and were labeled in accordance with what the decision from the common existing heuristic to replicate would have been (If the activity would have been chosen by the heuristic or not on the state of the JSSP from which data for the row was obtained). To obtain this data different JSSPs were solved with the common existing heuristics, and from each step information about the activities available for choosing was collected and labeled accordingly. The data collected was then used for training and testing the models.

## 5.1 Heuristics

The common existing heuristics used for this experiment were obtained from Garza et al. [19] and [20]. The hyper-heuristics were simple examples with random values. These were tested to see how the ML algorithms reacted to more complex decision making as players decision making is also expected to be complex. The heuristics and hyper-heuristics used are the following:

1. Shortest Processing Time (SPT): Select the activity that will take the shortest time to be done, in case of a tie choose the one with the shortest job id.

2. Longest Processing Time (LPT): Select the activity that will take the longest time to be done, in case of a tie choose the one with the shortest job id.

3. Maximum Job Remaining Time (MRT): Select the next available activity from the Job that has more pendant activities' time.

4. Most Loaded Machine (MLM): Compare the least loaded machines from each type and get the most loaded one from them. Get an activity that can be placed the earliest on the chosen machine, if there is no possible activity apply the same process with the rest of the machine types.

5. Least Loaded Machine (LLM): from each machine choose one that is the least loaded, then get the activity that can be placed the earliest on the chosen machine. If there is no possible activity apply the same process with the rest of the machine types.

6. Earliest Start Time (EST): choose the activity that can be placed the earliest on either machine, in case of a tie choose the one with the shortest job id.

7. Hyper Heuristic (HH) that chooses one of the previous six heuristics based on the Euclidian distance between the following features (described below on the feature section) which depended on the state of the problem: ATP, DPT, SLACK, DNPT and NATP; and a table of 5(features)X6(heuristics) with random values assigned to each cell; the heuristic row with the smallest Euclidian distance chose which heuristic to use.

8. Simple Hyper Heuristic (s-HH) 1-3: These where 3 hyper heuristics that each one depending of the value of two of these features: (ATP, DPT, SLACK, DNPT and NATP), selected between 4 of the common existing heuristics mentioned above. This meant that the heuristic used depended on the state of the problem.

## 5.2 Used Machine Learning (ML) Algorithms

The objective of this experiment is to find algorithms that can use data from a common existing heuristic solving the JSSP to create a model able to decide if an activity on a given state of a JSSP should be chosen, trying to replicate the behavior of the common existing heuristic used to create the data. The model should be able to receive a single vector with information about the activity and would give as an output a simple "yes" or "no" to know if the activity should be chosen. Six different ML techniques were used for training models with the data and are described below. These are all from the supervised area as the data is already labeled. Algorithms used can be seen in Table 5.1.

### 5.2.1 Decision tree

The decision tree is a tree-like graph (composed of nodes and branches) which helps to decide with which label to classify a row of data. Nodes from the tree lead you to a label and consist of decisions based on events (features and their values) as described by Magee [29]. For example, a node would have: "red color $<$ 100", this means that in the event where the red color has a value "x", this node leads you to left or right depending if its condition of x being less than a hundred is True or False. This process continues through the nodes until it finishes

on a label. The Decision Tree method was used since the model generated by the tree is easy to read which means that it has a better output for this thesis.

The decision tree algorithm used is the one from scikit learn called DecisionTreeClassifier. Many trees were made using all combinations with the following changes to the hyperparameters:

- Max_leaf_nodes: defines the maximum number of leaf nodes a node can have, it was set to 3 in all trees to establish a small limit for their width.
- Max_depth: defines a maximum depth for the tree, this was only tested with depth of 4 and depth of 3. The reason for this is that we want results to be easy to interpret and the longer the tree the more difficult it is to read. Also, heuristics should not need that much depth in a decision tree, which would mean that creating a longer tree than that would be overfitting the data.
- Criterion: Name of the function that would be used to measure the quality of the split. Both options were used ("gini" and "entropy"). Both gave similar answers but it helps to test options.
- Min_impurity_decrease: Defines a value that will give a restriction of how much impurity decrease does a split need to create to be transformed into a new node. The values tested are the default setting which is 0 and a value of 0.2.

### 5.2.2 K-Nearest Neighbors

This method takes all the test data already labeled as part of its model. It consists on determining $K$which will be the number of clusters and taking $K$ random points from the data to assign these points as the initial centers of the clusters. The $K$ means algorithm will take three steps until convergence as described by Ayodele [2]:

- Determine the center coordinate (the center of all points of the cluster)
- Determine the distance of each object to the center (Commonly using Eucledian Distance)
- Group the objects based on minimum distance (With $K$ centers)

After the $K$ clusters are created, every new point that gets to be classified calculates its distance to each of the $K$ centers and the one with the least distance determines the cluster where it belongs.

As Sutton [39] suggests, this method is used when the data points are separated into several different classes. This would mean that there should be a clear line across different dimensions (each feature being a dimension) where everything in the bound of that line is of a certain class. K-Nearest Neighbors algorithm was used to identify if the heuristics act over the data in an easily classifiable matter.

The K-Neighbors algorithm used is the one from scikit learn called KNeighborsClassifier. Many classifiers were made using different values for the parameter Nneighbors which defines the number of groups that are created (the number of neighbors). Values tested were 2, 3, 4, 5 and 8.

### 5.2.3 Multi-Layer Perceptron

Artificial neural networks are composed of a multitude of neurons representing elements that operate in parallel [17]. The Multi-Layer Perceptron is a type of neural network that uses the feed-forward method. It includes at least one hidden layer between the input and output layers and this makes it possible for the network to replicate non-linear functions. The reason this algorithm was used is because it is able to recognize different distributions better than the methods used for decision trees, which helps to test if there is a distribution in the features that a decision tree was not able to get but this network can.

The Neural Network algorithm used is the one from scikit learn called MLPClassifier, it uses log-loss as the squashing function. Many networks were made using all combinations with the following changes to the hyper-parameters and 350 max_iterations:

- Solver: defines a solver for the weight optimization change, solvers chosen where LBFGS and ADAM.
- Hidden_layer_sizes: defines length of the inner layers of neurons, each $n$ number placed is a new layer of $n$ nodes. The layers used are small (5,2), medium (10,10,10), mixed (20,5,20,5) and big (100,50,20,10,5).

### 5.2.4 Support Vector Machines

The SVM consists of mapping all points on a hyper plane and creating a plane with the dimensions equal to the hyper plane that best divides classes of the points. This means to create a plane that with the division that it makes, it maximizes the width between the points considering their categories. One class will be classified above the plane and the other one below it. Once the plane is placed, in order to predict the class of a new point, it is mapped on the hyper-plane and then its position relative to the plane is checked and used to define its class (over the plane means it is from one class and under for the other one). As the SVM uses a simple division for the data without affecting the points distribution, there is a technique in which kernels are used. Kernels are operations that are made over features of a point so the distribution of the points inside the hyperplane changes. This method could be compared with the K-neighbors method as both divide data using their position in the hyper-plane, but as it does this differently it is important to test both methods to increase variety. It could be important to mention that Ayodele [2] describes the method as being a close cousin to Multilayer Perceptrons.

The Support Vector Machine algorithm used is the one from scikit learn called SVM.SVC. Many Support Vector Machines were made using all combinations with the following changes to the hyper-parameters:

- Kernel: defines the kernel used. Options tested were: “rbf”, “linear”, and “poly”.

### 5.2.5 Random Forest

Random forest is a method that uses multiple decisions trees. Each decision tree is created individually using a random vector sampled independently as described by Breiman [6]. The

forest classifier ends up with many different trees (as the features each used for its creation were random) and to make a decision the mean/average of the result of each decision tree is obtained. Results from each decision tree have a value which depend on the accuracy of the individual tree on the data set from which the tree was made, and this value is used to define the importance of the decision given by that tree.

The random forest algorithm used is the one from scikit learn called RandomForestClassifier. Many classifiers were made using all combinations with the following changes to the hyper-parameters:

- N_estimators: this parameter defines the number of trees that the forest will have. Amounts tested were: 3, 5 and 10.
- Criterion: Name of the function that would be use to measure the quality of the splits in trees. Both options were used ("gini" and "entropy").
- Max_depth: defines a maximum depth for each tree, this was only tested with depth of 3. The reason for this is that we want results to be easy to interpret and the more depth the tree has, the more difficult it is to read, also heuristics should not need that long of a decision tree, which would mean that creating a longer tree than that would be overfitting the data.

| | |
|---|---|
| Tree with gini and max_depth 4 | TG4 |
| Tree with gini max_depth 4 and impurity decrease | TG4P |
| Tree with entropy and max_depth 4 | TE4 |
| Tree with entropy max_depth 4 and impurity decrease | TE4P |
| Tree with gini and max_depth 3. | TG3 |
| Tree with gini max_depth 3 and impurity decrease | TG3P |
| Tree with entropy and max_depth 3 | TE3 |
| Tree with entropy max_depth 3 and impurity decrease | TE3P |
| Small NN with lbfgs | SNNL |
| Medium NN with lbfgs | MNNL |
| Mixed NN with lbfgs | MxNNL |
| Big NN with lbfgs | BNNL |
| Small NN with adam | SNNA |
| Medium NN with adam | MNNA |
| Mixed NN with adam | MxNNA |
| Big NN with adam | BNNA |
| K-Neighbors (2 groups) | KN2 |
| K-Neighbors (3 groups) | KN3 |
| K-Neighbors (5 groups) | KN5 |
| K-Neighbors (8 groups) | KN8 |
| Linear Division SVC | SVC_L |
| Polynomial Division SVC | SVC_P |
| RBF Division SVC | SVC_RBF |
| Random Forest with gini (3 trees) | GRF3 |
| Random Forest with gini (5 trees) | GRF5 |
| Random Forest with gini (10 trees) | GRF10 |
| Random Forest with entropy (3 trees) | ERF3 |
| Random Forest with entropy (5 trees) | ERF5 |
| Random Forest with entropy (10 trees) | ERF10 |

Table 5.1: ML models used and their abbreviations.

## 5.3 Job-Shop Scheduling Problem (JSSP) instances

The JSSPs that were used are composed of $n$ jobs and $m$ machines, where each job has a $c$ number of activities. The constraints considered for the problems are the following:

1. Activities from a Job have an order, and that order must be respected as no activity can be scheduled to start before the other activities with higher precedence from the Job have been completed.

2. Machines can only process one activity at a time.

3. An activity that is scheduled to start on a machine must be completed on the same machine without interruption.

4. Activities and machines have an assigned type $t$, an activity can only be scheduled on a machine of the same type.

In order to create the JSS problems that were used for training and testing the ML algorithms, Taillard's method [41] was used. A random number generator, as seen in Table 5.2, was created. The algorithm receives a seed as input and outputs numbers between 0 and 1, these numbers are uniformly distributed because of the values of the constants 'a', 'b', 'c', and 'm'.

| | | |
|---|---|---|
| 0) | Initial seed and constants: | X0 (0 <X0 <231 - 1)<br>a = 16 807, b = 127 773, c = 2 836, m = 231 - 1 |
| 1) | Modification of the seed: | k := [Xi/b]<br>Xi+1 := a(Xi mod b) - kc<br>If Xi+1 <0 then let Xi+1 := Xi+1 + m |
| 2) | New value of the seed:<br><br>Current value of the generator: | Xi+1<br><br>Xi+1/m |

Table 5.2: Taillard's random number generator.

The algorithm for creating the instances is described in Algorithm 1 and it consists on getting as an input the amount of jobs, the amount of machines that were created and two seeds. In this algorithm $UNIF$ is the random number generator and floor is used to remove extra decimals. This algorithm works by generating one activity per machine for job, and assigning with the random number generator the time in which this activity needs to be completed (using the time seed). After that, each activity is assigned a machine, where first a number equal to the activity's position in the job represents the machine (Each job has the same number of activities as there are machines) and then a swapping function using the random number generator (and the machine seed) swaps the type (an activity can only be placed on a machine of the same type) between activities from the same job. An example of a JSSP obtained is shown in Table 5.3.

**Algorithm 1:** 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**

| | 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 |

Table 5.3: Example of a 3x3 JSSP instance.

## 5.4 Data composition

A JSSP has Jobs and Machines, each Job is composed of a number of activities. Each activity has two characteristics: a "complete time", which is the time it would take for a machine to complete the activity; and a "machine type" which is a number which identifies in which machine must the activity be done, as an activity can only be processed by a machine of the same type.The JSSP in Table 5.3 is composed of three jobs and each one has three activities. In each row of jobs there is a column named "machines" where values of "machine type" are presented and one named "times" which is for showing the "complete time" values of each activity. Each column has 3 values which are the values for each of the 3 activities from each job. The first value of column "machines" corresponds to the first value of the column "times", the second to the second one and so forth. One example is that in job 2 the first activity must be done in machine 2 and has a time size of 43.

Activities from each job were numbered from 0 to $n-1$, where $n$ is the number of activities in the job. This value will be called the "activity id", and it is unique between activities from the same job, but can be repeated between activities of different jobs. The "activity id" establishes the order in which activities from a job must be done, where an activity cannot start unless activities that have lower ids and are from the same job have already been completed. Activities in Table 5.3 are in order of the "activity id" where activities from column 1 of column "machines" start with an id of 0.

The JSSP was looked at in states (for better visualization a recommendation could be to see the graphical version of the JSSP shown on the solution model in Figure 3.2 from Chapter 3). The initial state for a JSSP is when none of the activities have been assigned to a machine. Activities can only be assigned to a machine of their same type in a time that complies with the restrictions that a machine can only be working on one activity at a time and activities from jobs must be done in order. Any valid combination of assigned and unassigned activities to machines is called a state. Each assignment of an activity to a machine is called a step and it always change the state of the JSSP. Unassigning activities from a machine would also be another step, but this was not be considered for this thesis.

In this thesis an activity cannot be assigned to a machine unless activities from the same job with higher precedence have already been assigned. Also, once activities have been assigned to a machine, their allocation cannot change. Because of this, only one activity from each job can be placed on each state. These activities must be the ones that have the lowest id of the activities that have not been assigned from each job and are called "available activities". In Table 5.3, on the beginning state the available activities are the first activities from each job. If the first activity of job 2 is assigned to a machine, the available activities will now be the first activities from job 1 and 3 but the second activity from job 2 as it is the next one to be assigned.

For each step (assignment of an activity to a machine) a set of data was generated, each set was composed of different rows conformed of the features defined. Each row contained information from an available activity of the step, and are labeled as True if the activity described was the one assigned to a machine in that step and False otherwise. So, in the dataset every row has information of the available activities on a state and a label to know if the activity was selected on said state, with the purpose of making Machine Learning methods learn which activities could be selected on each state of the JSSP.

### 5.4.1 Descriptive Features

Some features were made to uniquely identify each activity. Heuristics were used to define features that could be used by a ML algorithm to replicate the behavior of the heuristic. For example, for the heuristic "shortest processing time" there should be a feature that can uniquely identify if the activity is the shortest one from the available ones or not. This way it should be possible for the ML algorithms to label activities correctly by using features that directly connect to the heuristics. As activities are represented individually in each row and have no connection between the rows of other activities from the same step, some of the features were normalized between information of activities from the same step. Normalization is useful as it helps ML algorithms to understand the difference between the options available in each step of the JSSP, as now it is able to identify which of the activities had the lowest or highest

values on a feature. For example, for time size not normalized values from one step can be of 3, 35, 67, and from other step 5, 19, 33, but when normalized values from the first step mentioned will now be 0, 0.5, 1 and from the second one 0, 0.5, 1. With normalization the ML algorithm can know that the activities with the less costs are the ones with 0, and the ones with the highest cost are the ones with 1.

Defined features are explained below, for a better understanding, Tables 5.4 and 5.5 can be used as a guide.

1. Job id: Number of the activity's job. This value is normalized so ML algorithm could know if the activity chosen was from the first jobs available or the last ones.

2. Activity id: The id of the activity normalized, it can be helpful for identifying the progress of the activity's job against the others in terms on number of activities done (not in time values).

3. Cost: Time it takes to complete an activity normalized in two ways: one, with all the other of the possible activities to choose from (cost), and the other with the cost of all activities of the problem (total cost -TC). This feature is useful for recognizing SPT and LPT heuristics.

4. Time Left Job (TLJ): Total time of activities from the activity's job that have not been assigned. Feature used for the heuristic: Maximum Job Remaining Time.

5. Earliest time activity (ET): Earliest time an activity could be assigned, which is the end time of the previous activity from the same job.

6. Earliest possible time (EPT): Earliest time an activity can be assigned if placed at the end of any machine of its same type. This feature can be used to select activities as the EST heuristic.

7. Wasted time: considering EM is the lowest end time of the last activity placed on machines of the same type than the activity of the row, this feature would be the result of Earliest time minus EM. It will be the time wasted either because the activity could have been placed earlier but there was no machine available for that time (negative value); or because there was available position on a machine, but job restrictions did not permit the activity to be placed earlier (positive value).

8. Time taken by the machines of the activity's type (TTMT): the lowest end time of the last activity placed on machines of the same type than the activity of the row. It is a way of giving information on how the type of the activity is doing. This feature is useful for MRT, MLM and LLM heuristics.

9. Final Time: End time the activity would have if placed at the end of the machine. Calculated by summing the earliest possible time plus the activity's time cost.

Tables 5.4 and 5.5 show the data generated by three steps taken by the "Longest Processing Time" heuristic over the JSSP represented by Table 5.3. All features except Total Cost (TC) are normalized considering the values of the same step. On the other hand, TC has

| Step | Job Id | Activity Id | Cost | TC | ET | EPT | Wasted Time | TTMT | Label |
|---|---|---|---|---|---|---|---|---|---|
| 1 | 1 | 1 | 8 | 8 | 0 | 0 | 0 | 0 | FALSE |
| 1 | 2 | 1 | 43 | 43 | 0 | 0 | 0 | 0 | TRUE |
| 1 | 3 | 1 | 22 | 22 | 0 | 0 | 0 | 0 | FALSE |
| 2 | 1 | 1 | 8 | 8 | 0 | 0 | 0 | 0 | FALSE |
| 2 | 2 | 2 | 98 | 98 | 43 | 43 | 43 | 0 | TRUE |
| 2 | 3 | 1 | 22 | 22 | 0 | 0 | 0 | 0 | FALSE |
| 3 | 1 | 1 | 8 | 8 | 0 | 0 | 0 | 0 | FALSE |
| 3 | 2 | 3 | 1 | 1 | 141 | 141 | 141 | 0 | FALSE |
| 3 | 3 | 1 | 22 | 22 | 0 | 98 | -98 | 98 | TRUE |

Table 5.4: Example of data after three steps without normalizing.

| Step | Job Id | Activity Id | Cost | TC | ET | EPT | Wasted time | TTMT | Label |
|---|---|---|---|---|---|---|---|---|---|
| 1 | 0 | 0 | 0 | 0.0722 | 0 | 0 | 0 | 0 | FALSE |
| 1 | 0.5 | 0 | 1 | 0.433 | 0 | 0 | 0 | 0 | TRUE |
| 1 | 1 | 0 | 0.4 | 0.2165 | 0 | 0 | 0 | 0 | FALSE |
| 2 | 0 | 0 | 0 | 0.0722 | 0 | 0 | 0 | 0 | FALSE |
| 2 | 0.5 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | TRUE |
| 2 | 1 | 0 | 0.155 | 0.2165 | 0 | 0 | 0 | 0 | FALSE |
| 3 | 0 | 0 | 0.333 | 0.0722 | 0 | 0 | 0.695 | 0 | FALSE |
| 3 | 0.5 | 1 | 0 | 0 | 1 | 1 | 1 | 0 | FALSE |
| 3 | 1 | 0 | 1 | 0.2165 | 0 | 0.695 | 0 | 1 | TRUE |

Table 5.5: Example of normalized data after three steps.

been normalized considering all the time values from activities. In the normalized table it can be seen how activities labeled with true also have a Cost value of one, which is the feature and value that represented the ”Longest Processing Time” heuristic. This is how Machine Learning algorithms should be able to use feature values, and not only to make these simple decisions but also to later be able to recognize more complex patterns to understand human behavior.

### 5.4.2 State Features

The next list of features were obtained from the thesis written by Garza et al. [19] and [20]. These features were called ”state features” as they were used to describe the state of the JSSP and have the same purpose on this thesis.

Data that was used consisted on two different sets with different objectives; one of these sets had information “before movement” and the other one “after movement”. These names refer to the state from which ”state features” values were obtained. “Before movement” data has ”state features” in the state of the JSSP before the step (assignment of an activity) was applied. In the “after movement” data ”state features” are calculated in the state in which the JSSP would be if the activity that the row represents was assigned at its ”earliest possible time” (earliest time at the end of a machine). Considering that this data is used for training ML algorithms, the objective of a ML algorithm with “before movement” information would be to try to choose an activity given the current state of the problem. On the other hand, using the

“after movement” information would make the ML algorithm choose based on how assigning available activities change the state. Both methods were tested, as ”before movement” data requires less operations to obtain, but ”after movement” gives more unique information on the activity.

1. Average Processed Times (ATP): This feature is the percentage on how many activities have been processed by machines. It is calculated by obtaining the sum of all processing times for already processed activities divided by the sum of all processing times from all activities.

2. Dispersion of processing time index for scheduled activities (DPT): This feature is calculated by using processing times of already scheduled activities (activities which have been assigned to a machine) and calculating their standard deviation divided by their mean.

3. Percentage of slack in make-span (SLACK): Time from machine which is not being used to process activities, between initial time to makespan divided by total makespan of machines (the makespan is the maximum value from times taken by machines).

4. Dispersion of processing time index for pending activities (DNPT): This feature is calculated by using processing times of pending activities (activities which haven’t been assigned to a machine) and calculating their standard deviation divided by their mean.

5. Average Not Processed Times (NATP): This feature is the percentage on how many activities have not been processed by machines. It is calculated by obtaining the sum of all processing times for pending activities divided by the sum of all processing times from all activities.

6. Average pending processing time per job (NJT): To obtain this feature it is needed to sum all pending job’s times and divide by the total number of activities.

## 5.5 Application and Results

The experiments consisted of using the heuristics and hyper-heuristics defined in Section 5.1 to solve JSSPs and collecting data while the JSSPs were being solved. The JSS problems used were created using Taillard’s [41] generator as described in the JSSP instances section (Section 5.3). The input values for generating these problems are showed on Table 5.6. For each of the heuristics and hyper-heuristics a set of data was generated as described on the Data Composition section (section 5.4). For each of these sets one of each model defined by the ML algorithms described in the Machine Learning Algorithms section (Section 5.2) was trained using cross-validation of 4 splits with the data of each heuristic. This created multiple models where each of the models was trained using one of the possible ML algorithms and had the intention to emulate the behavior of one of the possible heuristics. Models are designed to receive a vector composed of the characteristics of a single activity in a state of a JSSP as an input, and to give as an output either True or False depending on if that particular activity on that specific state should be the next one to assign a time in a machine to, the accuracy of the model will depend if this output makes the same decision that the heuristic to replicate makes.

| # Jobs | # Machines | Time Seed | Machine Seed |
|---|---|---|---|
| 4 | 4 | 1166510396 | 164000672 |
| 4 | 4 | 840612802 | 398197754 |
| 4 | 4 | 1314640371 | 386720536 |
| 4 | 4 | 1227221349 | 316176388 |
| 5 | 4 | 533484900 | 317419073 |
| 5 | 4 | 1894307698 | 1474268163 |
| 5 | 4 | 874340513 | 509669280 |
| 3 | 3 | 1344106948 | 1868311537 |
| 3 | 3 | 425990073 | 1111853152 |
| 3 | 3 | 666128954 | 1750328066 |
| 4 | 4 | 342269428 | 1806358582 |
| 4 | 4 | 1603221416 | 1501949241 |
| 4 | 4 | 1357584978 | 1734077082 |
| 5 | 4 | 1124986343 | 1209573668 |
| 5 | 4 | 1463788335 | 529048107 |
| 5 | 4 | 1056908795 | 25321885 |
| 3 | 3 | 442723456 | 1369177184 |
| 3 | 3 | 2033800800 | 1344077538 |
| 3 | 3 | 964467313 | 1735817385 |

Table 5.6: Training problems.

### 5.5.1 Resulting Models

Results Tables 5.7 and 5.8 show the average obtained after applying the 4-splits cross validation to each of the models. These tables are color labeled, where the green color represents the higher percentages, the red color the lower percentages and yellow the ones in the middle. Values are compared between values of the same row (or values that come from evaluating the same heuristic). These colors can be helpful to visualize which algorithms did better as it can be seen with data in Tables 5.7 and 5.8, where Decision Trees (the columns that start with $T$ as "TG4" or "TE3") and Random Forests (the columns that start with $GR$) can be clearly identified to have more green values that other columns. With this information, it is possible to see how results do vary between algorithms, and to detect which are the better ones for the problem in hand.

As heuristics that were tested directly used at least one of the features to make a decision, on could have expected to see an accuracy of ($100\%$) with some of the models. This was not the case because ML algorithms consider each row from the data that was used as input as independent even if these were created in the same step, and in some cases knowing the information of the other activities was also necessary. For instance, if the heuristic being used is the "Longest Processing Time" (LPT) there could be a scenario where two activities have the same cost and are both the longest processing time. The activity selected in this scenario should be the one with the lowest job id, but as rows are independent and there is no feature that can identify the lowest id from two specific activities, ML algorithms are not able to label one of these rows correctly (The job id feature only identifies the lowest id from all available activities but cannot be used to compare between specific activities).

| H/ML | TG4 | TG4P | TE4 | TE4P | TG3 | TG3P | TE3 | TE3P |
|---|---|---|---|---|---|---|---|---|
| EST | 95.76% | 88.43% | 93.06% | 88.43% | 95.66% | 88.43% | 93.06% | 88.43% |
| SPT | 99.76% | 99.65% | 99.65% | 99.65% | 99.76% | 99.65% | 99.65% | 99.65% |
| LPT | 97.25% | 94.27% | 94.50% | 94.27% | 96.90% | 94.27% | 94.50% | 94.27% |
| MRT | 98.65% | 97.21% | 98.94% | 97.21% | 98.94% | 97.21% | 98.94% | 97.21% |
| MLM | 86.25% | 83.85% | 83.85% | 83.85% | 87.02% | 83.85% | 83.85% | 83.85% |
| LLM | 87.39% | 75.38% | 82.24% | 75.38% | 87.59% | 75.38% | 82.24% | 75.38% |
| HH | 83.77% | 69.86% | 73.13% | 69.86% | 77.34% | 69.86% | 73.13% | 69.86% |
| s-HH1 | 92.29% | 72.76% | 75.62% | 72.76% | 87.43% | 72.76% | 75.62% | 72.76% |
| s-HH2 | 98.50% | 90.22% | 94.36% | 94.36% | 97.70% | 90.22% | 94.36% | 94.36% |
| s-HH3 | 81.67% | 76.08% | 80.78% | 71.96% | 82.65% | 76.08% | 80.78% | 71.96% |
| Avg | 92.13% | 84.77% | 87.61% | 84.77% | 91.10% | 84.77% | 87.61% | 84.77% |

| H/ML | SNNL | MNNL | MxNNL | BNNL | SNNA | MNNA | MxNNA | BNNA |
|---|---|---|---|---|---|---|---|---|
| EST | 89.39% | 95.56% | 92.67% | 94.70% | 27.58% | 89.10% | 91.90% | 89.97% |
| SPT | 88.76% | 98.34% | 66.19% | 98.47% | 33.81% | 95.98% | 97.05% | 96.69% |
| LPT | 85.44% | 91.51% | 88.88% | 93.00% | 32.80% | 92.66% | 94.72% | 93.81% |
| MRT | 84.99% | 89.70% | 91.44% | 89.41% | 27.53% | 90.96% | 90.28% | 89.80% |
| MLM | 83.65% | 82.69% | 83.75% | 84.23% | 27.50% | 83.65% | 84.33% | 83.17% |
| LLM | 80.02% | 83.95% | 83.55% | 83.96% | 28.86% | 82.24% | 83.45% | 82.95% |
| HH | 73.44% | 76.61% | 69.34% | 71.97% | 30.14% | 75.66% | 70.18% | 74.29% |
| s-HH1 | 78.95% | 79.72% | 77.62% | 81.72% | 27.24% | 82.76% | 83.05% | 85.33% |
| s-HH2 | 81.23% | 89.52% | 87.91% | 87.91% | 32.91% | 89.06% | 90.10% | 88.26% |
| s-HH3 | 77.94% | 79.51% | 80.59% | 80.10% | 28.04% | 80.59% | 81.27% | 81.67% |
| Avg | 82.38% | 86.71% | 82.19% | 86.55% | 29.64% | 86.27% | 86.63% | 86.59% |

| H/ML | KN2 | KN3 | KN5 | KN8 | SVC_L | SVC_P | SVC_RBF |
|---|---|---|---|---|---|---|---|
| EST | 73.58% | 72.13% | 74.16% | 72.81% | 92.09% | 86.11% | 88.62% |
| SPT | 81.32% | 83.69% | 83.69% | 80.97% | 98.35% | 94.80% | 95.63% |
| LPT | 76.72% | 79.59% | 76.72% | 74.08% | 91.97% | 92.09% | 94.27% |
| MRT | 73.24% | 73.63% | 73.34% | 75.17% | 89.12% | 86.53% | 90.28% |
| MLM | 80.48% | 80.96% | 81.06% | 80.77% | 84.13% | 86.73% | 86.44% |
| LLM | 71.84% | 72.86% | 73.16% | 73.66% | 83.96% | 83.05% | 84.46% |
| HH | 72.60% | 67.65% | 68.39% | 70.71% | 71.44% | 77.56% | 79.03% |
| s-HH1 | 72.76% | 71.62% | 74.19% | 74.09% | 79.72% | 81.52% | 85.33% |
| s-HH2 | 74.91% | 75.95% | 74.22% | 72.73% | 87.34% | 87.80% | 90.79% |
| s-HH3 | 79.02% | 78.73% | 79.31% | 79.02% | 81.08% | 85.29% | 83.43% |
| Avg | 75.65% | 75.68% | 75.82% | 75.40% | 85.92% | 86.15% | 87.83% |

| H/ML | GRF3 | GRF5 | GRF10 | ERF3 | ERF5 | ERF10 | Avg |
|---|---|---|---|---|---|---|---|
| EST | 93.73% | 93.64% | 93.15% | 93.15% | 93.83% | 93.64% | 86.99% |
| SPT | 97.75% | 97.99% | 99.65% | 98.34% | 98.34% | 99.65% | 92.51% |
| LPT | 96.33% | 96.79% | 98.28% | 96.67% | 97.02% | 97.71% | 89.70% |
| MRT | 97.78% | 96.82% | 98.36% | 97.59% | 98.27% | 97.98% | 89.19% |
| MLM | 85.96% | 85.96% | 86.15% | 85.29% | 85.48% | 85.87% | 82.23% |
| LLM | 82.34% | 84.16% | 84.97% | 82.44% | 84.26% | 85.07% | 79.18% |
| HH | 74.39% | 74.39% | 74.61% | 75.24% | 75.45% | 75.03% | 71.89% |
| s-HH1 | 88.67% | 85.90% | 87.05% | 90.00% | 87.90% | 86.76% | 78.76% |
| s-HH2 | 97.82% | 97.36% | 96.89% | 97.93% | 97.01% | 97.24% | 87.90% |
| s-HH3 | 82.75% | 82.75% | 82.16% | 83.53% | 83.53% | 83.33% | 78.47% |
| Avg | 89.75% | 89.58% | 90.13% | 90.02% | 90.11% | 90.23% | 83.68% |

Table 5.7: Accuracy of each method using ”Before movement” information.

| H/ML | TG4 | TG4P | TE4 | TE4P | TG3 | TG3P | TE3 | TE3P |
|---|---|---|---|---|---|---|---|---|
| EST | 94.79% | 88.43% | 92.77% | 88.43% | 94.79% | 88.43% | 92.77% | 88.43% |
| SPT | 99.53% | 99.65% | 99.65% | 99.65% | 99.06% | 99.65% | 99.65% | 99.65% |
| LPT | 98.05% | 94.27% | 95.76% | 94.27% | 98.28% | 94.27% | 95.76% | 94.27% |
| MRT | 98.75% | 97.21% | 99.04% | 97.21% | 99.04% | 97.21% | 99.04% | 97.21% |
| MLM | 87.60% | 83.85% | 83.85% | 83.85% | 88.08% | 83.85% | 83.85% | 83.85% |
| LLM | 87.19% | 75.38% | 82.14% | 75.38% | 84.76% | 75.38% | 82.14% | 75.38% |
| HH | 79.45% | 69.86% | 72.92% | 69.86% | 74.08% | 69.86% | 72.92% | 69.86% |
| s-HH1 | 90.67% | 72.76% | 75.62% | 72.76% | 85.91% | 72.76% | 75.62% | 72.76% |
| s-HH2 | 97.58% | 92.06% | 94.36% | 94.36% | 96.89% | 92.06% | 94.36% | 94.36% |
| s-HH3 | 83.24% | 76.08% | 80.98% | 71.96% | 82.75% | 76.08% | 80.98% | 71.96% |
| Avg | 91.68% | 84.95% | 87.71% | 84.77% | 90.36% | 84.95% | 87.71% | 84.77% |

| H/ML | SNNL | MNNL | MxNNL | BNNL | SNNA | MNNA | MxNNA | BNNA |
|---|---|---|---|---|---|---|---|---|
| EST | 90.94% | 95.08% | 94.21% | 94.80% | 27.58% | 88.91% | 91.13% | 90.36% |
| SPT | 98.70% | 96.22% | 66.19% | 98.58% | 33.81% | 96.10% | 97.40% | 95.39% |
| LPT | 93.00% | 93.92% | 92.78% | 94.15% | 32.80% | 93.35% | 94.38% | 92.89% |
| MRT | 84.32% | 93.75% | 95.28% | 95.38% | 27.53% | 92.78% | 93.84% | 87.30% |
| MLM | 76.06% | 83.08% | 83.27% | 83.94% | 27.50% | 83.56% | 84.33% | 83.56% |
| LLM | 82.64% | 83.55% | 82.95% | 83.75% | 28.86% | 82.44% | 81.84% | 81.64% |
| HH | 70.92% | 73.23% | 71.12% | 71.65% | 30.14% | 75.03% | 71.55% | 75.34% |
| s-HH1 | 76.96% | 80.38% | 79.72% | 81.71% | 27.24% | 81.72% | 82.95% | 81.53% |
| s-HH2 | 90.45% | 90.33% | 89.64% | 88.72% | 32.91% | 90.45% | 90.44% | 91.02% |
| s-HH3 | 75.00% | 79.71% | 79.71% | 80.49% | 28.04% | 81.76% | 81.57% | 81.67% |
| Avg | 83.90% | 86.93% | 83.49% | 87.32% | 29.64% | 86.61% | 86.94% | 86.07% |

| H/ML | KN2 | KN3 | KN5 | KN8 | SVC_L | SVC_P | SVC_RBF |
|---|---|---|---|---|---|---|---|
| EST | 74.93% | 73.29% | 73.58% | 74.64% | 91.71% | 85.92% | 88.33% |
| SPT | 80.26% | 82.03% | 81.32% | 78.61% | 98.35% | 94.44% | 95.75% |
| LPT | 80.39% | 80.39% | 79.13% | 77.64% | 93.81% | 92.55% | 94.38% |
| MRT | 75.56% | 74.21% | 75.56% | 75.84% | 91.72% | 88.26% | 90.47% |
| MLM | 79.04% | 80.38% | 81.44% | 80.87% | 84.23% | 87.50% | 86.06% |
| LLM | 73.36% | 71.04% | 72.96% | 73.36% | 83.05% | 82.95% | 83.55% |
| HH | 70.18% | 67.23% | 69.23% | 71.55% | 71.76% | 75.98% | 76.92% |
| s-HH1 | 74.95% | 72.96% | 75.52% | 75.81% | 79.43% | 82.29% | 84.95% |
| s-HH2 | 76.64% | 75.72% | 75.38% | 74.57% | 88.61% | 88.61% | 91.37% |
| s-HH3 | 77.65% | 78.33% | 78.24% | 78.73% | 80.39% | 84.61% | 83.63% |
| Avg | 76.30% | 75.56% | 76.24% | 76.16% | 86.30% | 86.31% | 87.54% |

| H/ML | GRF3 | GRF5 | GRF10 | ERF3 | ERF5 | ERF10 | Avg |
|---|---|---|---|---|---|---|---|
| EST | 91.71% | 92.67% | 92.38% | 91.61% | 91.81% | 91.90% | 86.77% |
| SPT | 97.76% | 97.87% | 99.65% | 98.94% | 97.04% | 99.41% | 92.42% |
| LPT | 96.44% | 93.00% | 96.67% | 96.79% | 94.61% | 97.02% | 90.52% |
| MRT | 98.84% | 95.86% | 97.88% | 95.28% | 94.13% | 96.34% | 89.82% |
| MLM | 86.06% | 86.15% | 86.63% | 86.44% | 85.87% | 86.15% | 82.10% |
| LLM | 83.05% | 83.75% | 84.97% | 83.05% | 83.96% | 84.66% | 78.94% |
| HH | 73.23% | 74.18% | 74.29% | 73.76% | 74.39% | 74.29% | 71.20% |
| s-HH1 | 88.09% | 83.05% | 85.14% | 86.19% | 81.72% | 84.38% | 78.12% |
| s-HH2 | 94.71% | 91.37% | 95.05% | 95.40% | 94.94% | 95.97% | 88.22% |
| s-HH3 | 82.75% | 82.75% | 83.33% | 82.16% | 82.55% | 83.82% | 78.31% |
| Avg | 89.26% | 88.07% | 89.60% | 88.96% | 88.10% | 89.40% | 83.64% |

Table 5.8: Accuracy of each method using ”After movement” information.

### 5.5.2 Results from Applying the ML Algorithms

Results from Subsection 5.5.1 tested accuracy considering only how many activities were labeled correctly. These values might be useful for evaluating the machine learning algorithms, but might be misleading as most of the activities are labeled as False (around two thirds). Because of this, a test was made in which the models created were used to choose a single activity for each step of the JSSP. Most steps of the JSSP have multiple activities from which to choose, and the models can only label each one of the activities individually as True or False, so it can happen that on the same step more than one activity is labeled as True or that all activities are labeled as False. So, this experiment tested each of the models (the one with the highest accuracy from the Cross-validation) by taking as their selection the first activity (the one with the lowest job id) that was labeled as True, and if all of them were labeled as False then its selection would be the first activity of the available ones. Using all the JSSPs generated in this section each of the models was compared with the heuristics to imitate on their selection of an activity on each of the steps when solving each of the JSSPs. The accuracy represents how often does the model select the same activity than the heuristic.

Results are shown on Tables 5.9 and 5.10, where it can be seen that for some models the accuracy decreased but for others a perfect accuracy was obtained. This is because now whenever there was a tie between two values on the main feature, the selection method used for this experiment selected the activity that the heuristic would have also chosen. This is the case for the Shortest Activity First heuristic for example, as if the two lowest cost activities have the same size, the heuristic would choose the one with the lowest job. On the other hand, the Most Loaded Machine heuristic, whenever two machines are equally loaded, it would choose the activity that can be placed the earliest on that machine, but ML algorithms were not able to replicate this behavior. Another interesting thing to notice on the data is that there is a difference between the results of the random forest between “Before Movement” and “After Movement” data on the simple hyper-heuristics data, this might be due to the fact that the simple hyper-heuristics used “Before Movement” state information to make decisions, but decision trees did not seem to have the same problem.

| H/ML | TG4 | TG4P | TE4 | TE4P | TG3 | TG3P | TE3 | TE3P |
|---|---|---|---|---|---|---|---|---|
| EST | 85.66% | 100.00% | 83.22% | 100.00% | 86.36% | 100.00% | 83.22% | 100.00% |
| SPT | 100.00% | 100.00% | 100.00% | 100.00% | 100.00% | 100.00% | 100.00% | 100.00% |
| LPT | 98.95% | 100.00% | 95.45% | 100.00% | 100.00% | 100.00% | 95.45% | 100.00% |
| MRT | 100.00% | 100.00% | 100.00% | 100.00% | 100.00% | 100.00% | 100.00% | 100.00% |
| MLM | 69.93% | 70.98% | 70.98% | 70.98% | 67.13% | 70.98% | 70.98% | 70.98% |
| LLM | 77.27% | 74.48% | 73.08% | 74.48% | 77.27% | 74.48% | 73.08% | 74.48% |
| HH | 73.43% | 40.21% | 40.21% | 40.21% | 64.69% | 40.21% | 40.21% | 40.21% |
| s-HH1 | 91.96% | 33.22% | 76.22% | 33.22% | 83.92% | 33.22% | 76.22% | 33.22% |
| s-HH2 | 100.00% | 94.41% | 94.06% | 94.06% | 97.55% | 94.41% | 94.06% | 94.06% |
| s-HH3 | 75.52% | 37.06% | 69.23% | 37.06% | 74.83% | 37.06% | 69.23% | 37.06% |
| Avg | 87.27% | 75.03% | 80.24% | 75.00% | 85.17% | 75.03% | 80.24% | 75.00% |

| H/ML | SNNL | MNNL | MxNNL | BNNL | SNNA | MNNA | MxNNA | BNNA |
|---|---|---|---|---|---|---|---|---|
| EST | 93.36% | 94.06% | 91.61% | 88.81% | 36.36% | 69.58% | 90.56% | 79.72% |
| SPT | 44.06% | 92.31% | 44.06% | 99.30% | 44.06% | 91.96% | 96.85% | 93.36% |
| LPT | 87.06% | 86.71% | 88.11% | 87.41% | 48.95% | 89.16% | 95.80% | 89.51% |
| MRT | 75.52% | 86.36% | 89.16% | 80.42% | 33.57% | 86.71% | 75.87% | 84.97% |
| MLM | 66.08% | 62.94% | 67.48% | 63.99% | 33.22% | 70.63% | 69.93% | 70.28% |
| LLM | 70.28% | 70.28% | 73.43% | 71.68% | 41.26% | 70.28% | 75.17% | 74.48% |
| HH | 40.21% | 45.80% | 48.25% | 47.90% | 40.21% | 48.95% | 40.56% | 39.51% |
| s-HH1 | 54.20% | 52.80% | 52.45% | 55.59% | 33.22% | 56.64% | 63.64% | 61.54% |
| s-HH2 | 81.47% | 88.46% | 82.87% | 84.97% | 47.55% | 87.06% | 84.27% | 83.92% |
| s-HH3 | 60.49% | 61.54% | 63.99% | 59.44% | 37.06% | 64.34% | 68.88% | 65.03% |
| Avg | 67.27% | 74.13% | 70.14% | 73.95% | 39.55% | 73.53% | 76.15% | 74.23% |

| H/ML | KN2 | KN3 | KN5 | KN8 | SVC_L | SVC_P | SVC_RBF |
|---|---|---|---|---|---|---|---|
| EST | 57.34% | 65.73% | 59.09% | 42.66% | 82.52% | 68.53% | 81.82% |
| SPT | 75.87% | 81.12% | 78.32% | 73.43% | 98.25% | 95.10% | 96.85% |
| LPT | 75.87% | 83.22% | 76.92% | 65.38% | 87.06% | 87.06% | 92.66% |
| MRT | 55.24% | 65.38% | 50.70% | 42.66% | 81.82% | 78.32% | 90.91% |
| MLM | 68.18% | 74.48% | 69.58% | 62.24% | 65.38% | 77.27% | 76.92% |
| LLM | 61.89% | 69.23% | 62.24% | 58.74% | 70.63% | 66.43% | 73.43% |
| HH | 55.24% | 62.94% | 54.20% | 44.76% | 40.91% | 59.09% | 62.59% |
| s-HH1 | 52.10% | 61.89% | 55.94% | 41.96% | 50.70% | 59.09% | 73.08% |
| s-HH2 | 73.43% | 79.72% | 73.43% | 62.24% | 81.82% | 83.57% | 88.81% |
| s-HH3 | 67.48% | 73.78% | 67.48% | 59.44% | 65.03% | 75.17% | 73.43% |
| Avg | 64.27% | 71.75% | 64.79% | 55.35% | 72.41% | 74.97% | 81.05% |

| H/ML | GRF3 | GRF5 | GRF10 | ERF3 | ERF5 | ERF10 | Avg |
|---|---|---|---|---|---|---|---|
| EST | 93.01% | 94.76% | 95.45% | 89.51% | 94.76% | 95.10% | 82.86% |
| SPT | 91.96% | 91.61% | 100.00% | 90.91% | 96.85% | 100.00% | 88.84% |
| LPT | 98.60% | 98.95% | 98.95% | 99.65% | 100.00% | 98.60% | 90.54% |
| MRT | 93.01% | 85.31% | 97.55% | 95.80% | 96.50% | 97.90% | 84.27% |
| MLM | 74.48% | 73.78% | 73.43% | 71.33% | 73.43% | 72.73% | 68.99% |
| LLM | 75.17% | 76.92% | 74.83% | 75.17% | 76.92% | 77.97% | 71.21% |
| HH | 44.06% | 45.45% | 46.50% | 67.13% | 50.00% | 49.30% | 48.72% |
| s-HH1 | 83.57% | 78.67% | 70.63% | 88.46% | 78.67% | 70.28% | 60.56% |
| s-HH2 | 99.65% | 99.30% | 98.95% | 99.65% | 99.30% | 97.90% | 87.62% |
| s-HH3 | 70.63% | 72.38% | 69.58% | 72.03% | 72.38% | 72.38% | 63.07% |
| Avg | 82.41% | 81.71% | 82.59% | 84.97% | 83.88% | 83.22% | 74.67% |

Table 5.9: Accuracy of each method using ”Before movement” information with applied results.

| H/ML | TG4 | TG4P | TE4 | TE4P | TG3 | TG3P | TE3 | TE3P |
|---|---|---|---|---|---|---|---|---|
| EST | 82.87% | 100.00% | 83.57% | 100.00% | 82.87% | 100.00% | 83.57% | 100.00% |
| SPT | 99.65% | 100.00% | 100.00% | 100.00% | 99.65% | 100.00% | 100.00% | 100.00% |
| LPT | 100.00% | 100.00% | 99.30% | 100.00% | 100.00% | 100.00% | 99.30% | 100.00% |
| MRT | 99.65% | 100.00% | 100.00% | 100.00% | 100.00% | 100.00% | 100.00% | 100.00% |
| MLM | 72.73% | 70.98% | 70.98% | 70.98% | 73.08% | 70.98% | 70.98% | 70.98% |
| LLM | 70.98% | 74.48% | 74.83% | 74.48% | 69.23% | 74.48% | 74.83% | 74.48% |
| HH | 75.87% | 40.21% | 43.71% | 40.21% | 58.74% | 40.21% | 43.71% | 40.21% |
| s-HH1 | 87.76% | 33.22% | 76.22% | 33.22% | 80.42% | 33.22% | 76.22% | 33.22% |
| s-HH2 | 98.95% | 94.41% | 92.66% | 92.66% | 98.95% | 94.41% | 92.66% | 92.66% |
| s-HH3 | 77.62% | 37.06% | 69.23% | 37.06% | 75.87% | 37.06% | 69.23% | 37.06% |
| Avg | 86.61% | 75.03% | 81.05% | 74.86% | 83.88% | 75.03% | 81.05% | 74.86% |

| H/ML | SNNL | MNNL | MxNNL | BNNL | SNNA | MNNA | MxNNA | BNNA |
|---|---|---|---|---|---|---|---|---|
| EST | 73.78% | 92.31% | 91.96% | 91.26% | 36.36% | 72.03% | 71.68% | 77.97% |
| SPT | 98.25% | 98.60% | 44.06% | 98.95% | 44.06% | 94.41% | 96.15% | 92.66% |
| LPT | 95.10% | 90.56% | 87.41% | 91.26% | 48.95% | 91.26% | 94.41% | 91.26% |
| MRT | 96.50% | 93.36% | 91.61% | 93.36% | 33.57% | 89.86% | 92.31% | 80.07% |
| MLM | 33.22% | 63.29% | 64.34% | 64.69% | 33.22% | 62.24% | 70.98% | 60.49% |
| LLM | 66.08% | 67.83% | 66.78% | 67.83% | 41.26% | 69.58% | 65.38% | 57.69% |
| HH | 43.01% | 43.71% | 40.21% | 46.85% | 40.21% | 50.70% | 40.91% | 40.91% |
| s-HH1 | 57.34% | 56.99% | 59.79% | 59.79% | 33.22% | 62.94% | 64.34% | 70.63% |
| s-HH2 | 85.31% | 85.31% | 86.01% | 82.52% | 47.55% | 87.41% | 88.81% | 93.01% |
| s-HH3 | 37.41% | 62.59% | 68.18% | 63.29% | 37.06% | 67.13% | 67.83% | 67.13% |
| Avg | 68.60% | 75.45% | 70.03% | 75.98% | 39.55% | 74.76% | 75.28% | 73.18% |

| H/ML | KN2 | KN3 | KN5 | KN8 | SVC_L | SVC_P | SVC_RBF |
|---|---|---|---|---|---|---|---|
| EST | 58.39% | 68.88% | 62.24% | 49.65% | 84.62% | 70.98% | 79.02% |
| SPT | 79.37% | 83.22% | 79.37% | 70.28% | 98.25% | 93.71% | 96.50% |
| LPT | 74.48% | 79.37% | 75.52% | 69.23% | 92.66% | 89.86% | 93.36% |
| MRT | 55.24% | 67.13% | 55.59% | 40.21% | 86.01% | 80.07% | 90.91% |
| MLM | 63.29% | 69.93% | 65.03% | 61.54% | 65.03% | 76.22% | 74.48% |
| LLM | 68.18% | 73.43% | 66.08% | 56.64% | 68.53% | 66.08% | 72.03% |
| HH | 58.04% | 60.14% | 55.59% | 48.25% | 40.21% | 55.94% | 57.34% |
| s-HH1 | 55.59% | 62.59% | 54.90% | 45.10% | 53.15% | 59.79% | 73.78% |
| s-HH2 | 71.33% | 75.17% | 72.03% | 66.08% | 82.87% | 84.97% | 89.51% |
| s-HH3 | 63.99% | 69.23% | 66.08% | 61.54% | 63.64% | 73.43% | 72.73% |
| Avg | 64.79% | 70.91% | 65.24% | 56.85% | 73.50% | 75.10% | 79.97% |

| H/ML | GRF3 | GRF5 | GRF10 | ERF3 | ERF5 | ERF10 | Avg |
|---|---|---|---|---|---|---|---|
| EST | 89.16% | 91.96% | 94.76% | 89.16% | 91.96% | 95.10% | 81.59% |
| SPT | 98.95% | 94.76% | 100.00% | 98.60% | 93.01% | 100.00% | 91.46% |
| LPT | 99.30% | 95.80% | 98.60% | 99.30% | 96.15% | 98.60% | 91.42% |
| MRT | 98.60% | 89.86% | 98.60% | 92.66% | 82.87% | 94.06% | 86.28% |
| MLM | 72.73% | 72.73% | 73.43% | 71.33% | 71.33% | 71.33% | 66.64% |
| LLM | 67.83% | 70.28% | 74.48% | 67.83% | 70.28% | 74.83% | 68.51% |
| HH | 46.50% | 46.85% | 46.50% | 46.15% | 46.15% | 46.15% | 47.70% |
| s-HH1 | 64.69% | 58.39% | 57.69% | 60.49% | 57.69% | 65.03% | 58.19% |
| s-HH2 | 95.80% | 87.76% | 93.01% | 95.80% | 96.50% | 96.15% | 86.91% |
| s-HH3 | 66.08% | 68.88% | 70.28% | 67.13% | 67.48% | 71.33% | 61.82% |
| Avg | 79.97% | 77.73% | 80.73% | 78.85% | 77.34% | 81.26% | 74.05% |

Table 5.10: Accuracy of each method using ”After movement” information with applied results.

### 5.5.3 Conclusions

Data showed how decision trees with impurity decrease did better on the common existing heuristics than the ones without it, but when it came to more complex situations as the simple hyper-heuristics, trees with impurity decrease did badly. This happened because trees without impurity decrease over-fitted with the common existing heuristics but because of the same thing, when it came to the hyper-heuristics, they were able to detect more information. Because of the data generated in this chapter it is possible to select from the ML algorithms tested the ones that are better for this project. The three ML algorithms selected were: Decision Trees, SVMs and Random Forests as these were the ones with higher values in their accuracy percentages. From Decision Trees only the ones without the impurity decrease as these did better with the hyper heuristics, from the SVMs the Polynomial and the RBF divisions were selected as these two had interesting accuracies and because of their kernels data is handled differently than with decision trees. And from the random forests the ones composed by 3 trees and by 10 trees. giving a total of 10 selected ML algorithms.

## 5.6 Summary

In this chapter, ML algorithms were tested to see if these were capable of replicating heuristics. For this JSSPs were solved using basic heuristics obtained from the thesis of Garza et al. [19] and [20], and some hyper heuristics created. While the heuristics and hyper heuristics were solving the problem, from each movement done, data of the state of the JSSP was being collected. The data was composed by the features that were defined based on the the same heuristics together with some state features which also came from the thesis of Garza et al. [19] and [20]. This data was then used to train and test different ML algorithms to see if these were able to accurately choose the same options as heuristics. Good results were obtained, Decision Trees, SVMs and Random Forests were selected as the main algorithms to use in this thesis, and it was concluded that some ML algorithms are capable of replicating heuristics.

# Chapter 6

# Developing the Video Game

This chapter includes the process of building the video game that was used to collect the data from humans solving JSS problems. Using a video game was a decision made with the intention of making people solve problems willingly by giving them entertainment in exchange for their time. The research from Morschheuser et al. [32] shows that gamifying a problem also increases engagement and quality of results, which is of high value for this research as better results can lead to better heuristics.

After analyzing the objective of building the video game, some characteristics were defined before its making, these were:

- It must be of easy access. This game was distributed among many people with little background consideration, the only expectations are for them to have common computer handling abilities and common problem-solving capabilities. Given this, complications for accessing and interacting with the game were planned to be reduced, which included avoiding installation of software and giving easy login access. This was done by making the game runnable on a web page so it could be accessed and used in most browsers, and making the log-in to only require a username and a password which is a small amount of information.

- It must be understandable. The JSSP is a complex problem, and it can be complicated to explain it to players. For the purpose of getting people to get good solutions for the JSS problems it is of high priority for them to understand the problem correctly. To accomplish understandability two main steps were done when making the video game. The first one was to test out a beta-version of the game where the difficulties that humans encountered while solving the JSSPs were analyzed. After learning what confused people visual cues were added in the game to help future players avoid these difficulties. The second step was to build a tutorial where a step by step guide on how to solve the JSS problem in the game was explained, including errors you can make and all type of moves that can be done.

- It must gather enough data to replicate human behavior. Even if the features that were used for this project were defined in the last chapter, being able to replicate the movements that the player made when playing instead of getting the needed features directly has a higher value. This is because if there is any change in the features needed there

will be no need to collect new data from users, but only to replicate their movements and collect the data with the new features. This was achieved by collecting information of the "when", "what" and "where" from any movement of an activity that the player did.

## 6.1 Designing the Video Game

The process followed for designing the video game started with establishing how the properties of the JSS problem would be used in the game. The main objective consisted of creating a game in which actions done while playing it could be transformed and used to get a solution for a JSSP. This can be done in many ways; a simple method would be to create a game by applying simple modifications to JSSP concepts transforming them into game elements without removing the important rules that make the JSSP its own problem. This simple method is easy to understand, it pushes to transform the elements of the JSSP into elements of a game, making them move similarly and with the same set of rules than those of the JSSP. A second option to complete the objective is to solve pieces of the JSSP throughout the game without worrying about any order or a direct translation of the elements in the game to JSSP components.

A quick comparison of the two methods is to consider for the simple one a game where players must manage a virtual company and for that they need to solve a JSSP for the scheduling of their in-game business. The JSSP is given to the player with a graphical representation of it and the player solves the problem directly to gain points. For the second method, consider an adventure/shooting game where decisions made by the player in the adventure, like where to go first or what to do next, can be translated into movements of what should a next move be at the beginning of the JSSP. This can be done if making these decisions has similar or the same restrictions and characteristics as those that a JSSP has at its beginning. Then, decisions made in the shooting section where the player must shoot different types of targets with different types of gun depending on the type, could be used to translate them into decisions of what movement to do at the end of the JSSP. An interesting consideration is that in this game the player might not know that a JSSP can be solved with the decisions that are being made as it is not shown directly.

Both methods are relevant for projects with the objective of getting humans to solve a problem in the form of a video game. The simple method is the easiest and fastest to work with, but the second method can open up more possibilities of what the game could be, and it could even lead to finding out a current popular game that is already making players solve the problem in hand. Because of this, the more complex method has a greater potential of getting the gathering of heuristics from video game data into the commercial world rather than just keeping it in the experimentation and study field.

The definition for the JSSP includes jobs composed of activities that must be done in machines. This definition was simplified to obtain the main characteristics of the JSSP, which is: the JSSP includes elements with a size that must be "done" by something taking up a number of units proportional to the size of the elements to be done. Restrictions like the order in which elements must be done, how the types of the elements define what has to do them, or that the "somethings" cannot be doing more than one element at a time are also part of the

generalized definition. Even if the new definition looks like a more informal and subjective version of the original one, it is important to take away the limitations that the words gave. Now that there is not a specification that there are machines completing activities, the activities can now be viewed as for example “mining ore” where the time an activity took to complete can now be represented by the size of the ore, and instead of using machines “miners that have to mine the ore” can be used. This helps to expand the possible elements that can constitute the game, which is a relevant step for transforming the JSSP into a game as it helps to imagine different possibilities for game design.

### 6.1.1 Graphical JSSP

The simplest method to transform the JSSP into a game is to take a graphical representation of it, add its rules to the movements of each object, and gamify it. An option for a graphical representation of the JSSP can be composed of only rectangles of different colors and outlines, composed of big rectangles with no inner color to represent machines, and smaller rectangles to represent activities as seen in Figure 6.1. The inner color of the activity rectangles represents their job. This means that all activities from the same job will have the same color. The outline of both the activity rectangles and the machine rectangles represent their type. In the JSSP activities can only be done by machines of their type, so in this case activity rectangles can only be placed on machine rectangles with the same outline color.

The width of all rectangles can be the same (machine rectangles are a little wider so activity rectangles can fit inside them), but the length of activity rectangles were used to represent the time it takes to complete the activity they correspond to. Machine rectangles could also have a length which could be a maximum amount of time that the player will have to complete all activities, this can be the time of the optimal solution, or an average time, etc.

In this representation the horizontal or x-axis is being used to represent time, this should be considered while complying with the JSSP restrictions. These include that activities of the same job must be done in order, this would mean that the end of a previous activity must be at the left of the start of the next activity considering their “x-axis” (see in Figure 6.2); also machines can only do one activity at once, which would mean that activity rectangles cannot overlap.

The design just described was the one used when creating the video game. Other designs were considered but it was decided that this one was the easiest to build. It can also be argued that by showing players a direct representation of the JSSP, the strategies generated by them will be directed to solving JSS problems, which can be beneficial for this thesis. In this chapter other designs are described even if these were not applied. As the information that was gained by their creation can be helpful for further study.

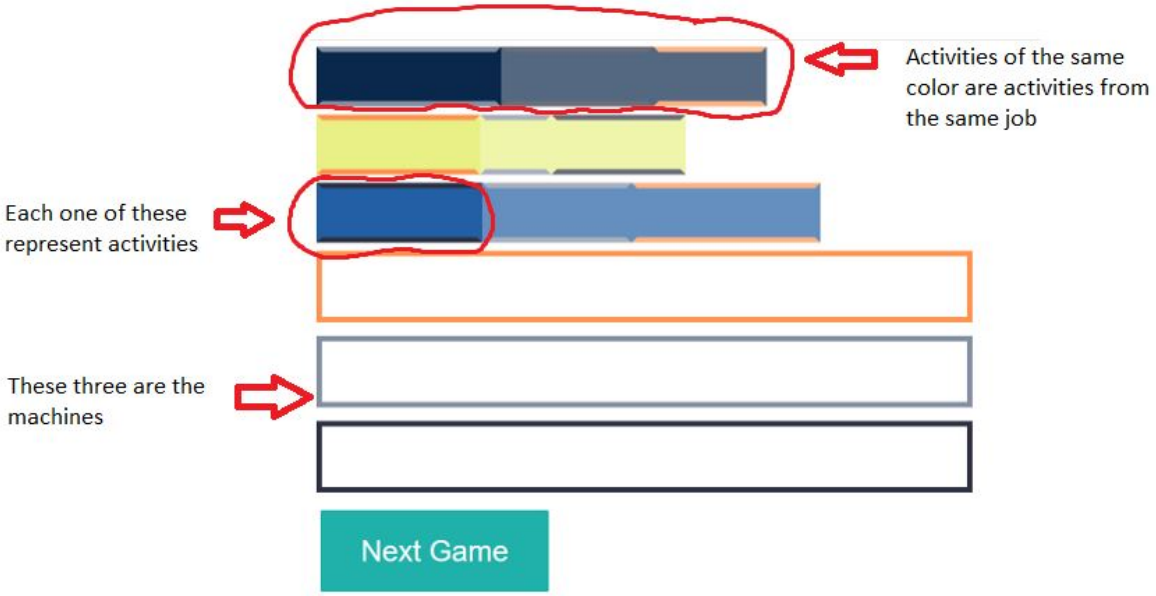


Figure 6.1: Graphical JSSP.

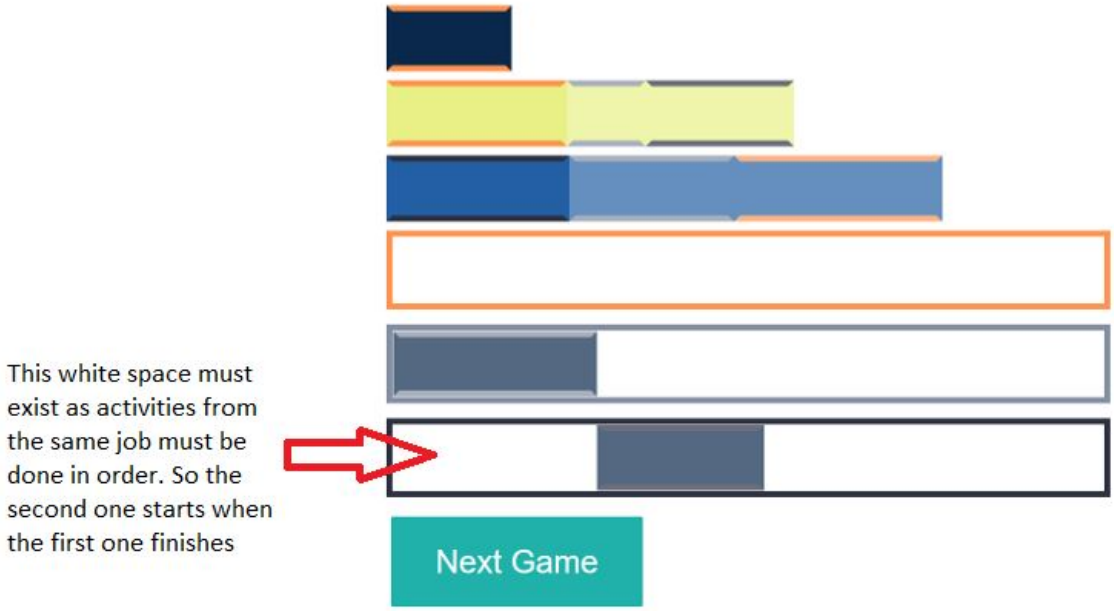


Figure 6.2: Graphical JSSP Job Restriction.

### 6.1.2 4D-Tetris

One of the most relevant designs created was called 4D-Tetris. This was an idea inspired from the original Tetris game but does not use most of the relevant characteristics from the original game. While in the original Tetris there is an area where blocks of different shapes and sizes fall and are placed one over another, in this version there is more than one area, each one representing a machine of a JSSP. Pieces are representing activities and only have one shape, the stick form, all with the same width but varying in length depending on their time values. These pieces are called activities to make the description of the game more understandable. See Figure 6.3 for reference. An easy way to understand this transformation is to view it as solving the JSSP sideways.

Each activity that falls on the same area than other activities will fall over them and cannot go through them. This will always be true, but an activity will not always fall exactly on top of the upmost activity on the area as there are other restrictions to also consider. The lowest an activity can go on an area is called its final position. When an activity starts to fall, before it touches its final position on the area, there is the option to change it and try to place another available activity instead (available activities are from each job the first activity of the ones that haven't been placed yet in one of the fall areas). This can be seen in Figure 6.3 with the blue arrow.

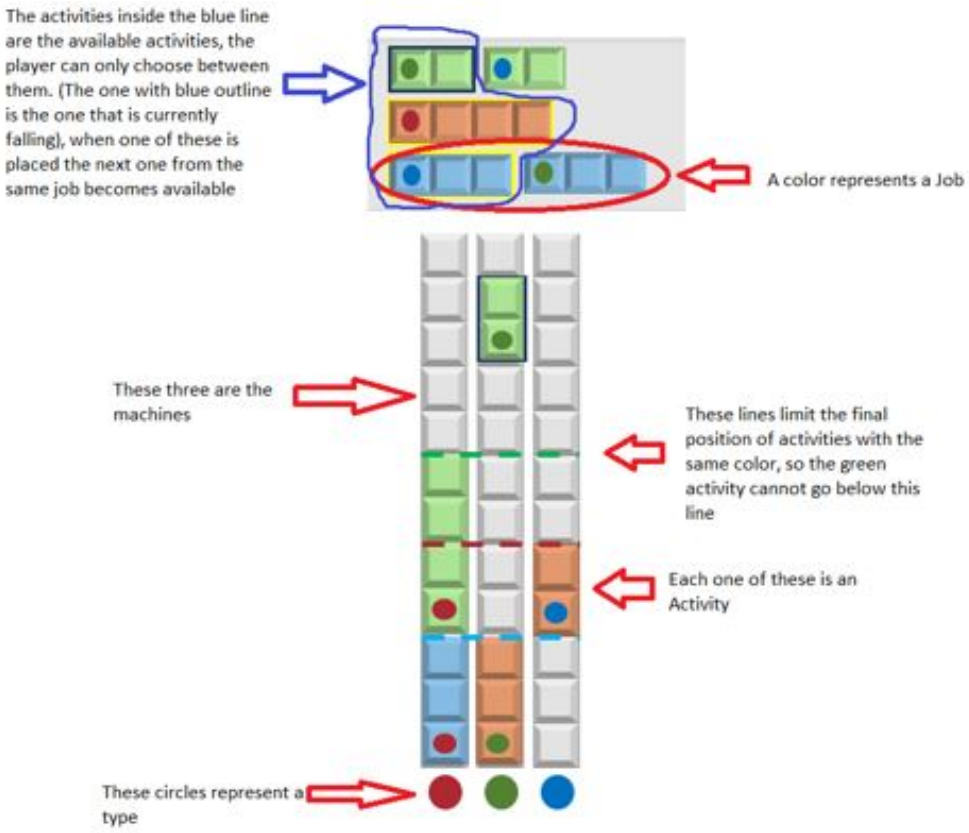


Figure 6.3: 4D-Tetris.

When an activity is falling on an area its final position does not only depend on the activities below, but also on the limits set by other activities from the same job, where each activity from each job will create a line of its color on all areas and any activity of the same color cannot go below the line generated. (This can be seen on Figure 6.3)

In the JSSP a rule of types exist, where an activity of a type can only be done by a machine of the same type. As areas represent machines it is important to mention that activities should only automatically appear over areas of its type. All fall areas have an upper limit, if any of the activities are above the limit while being on their final position then the game ends. The final score should be relative to the number of activities placed on their area.

This game was not made as it was considered that the fall-time limit could become distracting for players making them lose focus on the main objective which is to solve the JSSP. Still, in comparison with the game design chosen, this one was more likely to force the player to think on a heuristic kind of way by making them make quick decisions using only the next available activities and placing them on top of activities that were already placed.

### 6.1.3 Monster Hunters

With the intention of making a more typical game, another idea that was explored for the design was a multiplayer game that consisted of each player completing different missions which could also be viewed as multiple minigames (in this case these minigames would be attacking monsters). The JSSP is manifested as players can be used to represent machines and each mission could represent the activities.

The game starts with a room with different doors, each door can have a specific monster which has to be attacked multiple times, each time by a specific type of player. Each monster represents a job, defeating a monster takes multiple steps (the missions) where each one represents an activity, this shows a job that consists of different activities of the JSSP. This can be seen in Figure 6.4. These steps should be completed by specific types of players just like activities must be done by specific machines.

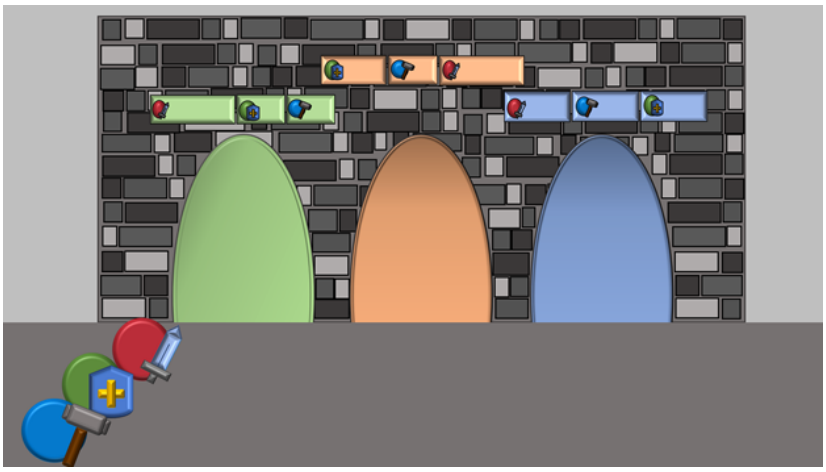

Figure 6.4: Monster Hunters.

At the start of the game each player can go to any of the doors that need a player of their type to complete the missions. There must be a way for players to know which steps are needed to defeat each monster and what type of player must do it. This is important as it is how players would interact with the JSS problem. There should be a time limit for defeating all the monsters, and the more the monsters are attacked the more points players get. This would encourage players to form a strategy on the best order to attack the monsters.

Each mission for each monster should take some time similar to the activity it represents, the mission does not have to last the exact amount of time than that of the activity, but it must be relatively similar. This means that the time for completion of a mission can be dependent on the skill of the player. However, the average time for completion must be representative of the time of the activity this mission represents relative to the other missions/activities in the game.

As there will be times where players will not have a mission of their type available, activities without any time limitations can be added so these players can keep playing while waiting for their teammates. These activities could give extra points, but attacking monsters need to give enough points to motivate the players to focus on attacking the monsters rather than completing the extra activities.

There is a game called League of Legends (LoL) which was developed and published by Riot Games[18]. An oversimplified description of it is that it consists on a multiplayer versus game where players must destroy the main tower of the opposing group of players. There are different towers of each team that can also be destroyed by the opposing team, which is done as these generate attacking minions that attack the enemy. The relevant part of this game to this section is that there are also monsters that can be defeated to gain buffs. Another relevant fact is that players choose characters that are of different types. The main idea here is that this game and monster hunters (the game that was being described for the JSSP) could be combined. This could be done by adding the JSSP rules that were added to the monsters in monster hunters to LoL monsters. In the actual game, LoL's main objective is to defeat the enemy tower and not really to defeat the monsters, also the way players play the game gives each type of player a specific objective when playing (one defends the top, other the center etc). To make the players focus on the monsters, buffs obtained by defeating them must be of high significance to the game. The official release of LoL was on 2009 and it is still a relevant and popular game to this day, this game is also played competitively which means that players strive to be good at it. The fact that there is a way to transform or insert a typical optimization problem in computer science into a well known video game (the JSSP into LoL) shows the potential of being able to use game data to obtain new heuristics.

This solution is really interesting and of high value, but its making requires a high amount of resources (time, players, money) that are not available for this thesis, which is why this design was not implemented.

## 6.2 First Version of the Video Game

A first version of the game was created for testing it out with different people. The objective was to see how humans interacted with the JSS problem and knowing if there were any complications on understanding the game and its interactions. This first version included the options of user creation and saving capacity together with the gamified graphical JSSP. It was also capable of collecting data of players' movements.

The graphical representation of the JSSP described above was used in this version. It is the one in which activities are represented by colored rectangles where their inner color stands for the job these belong to and their outline color illustrates the machine these go into.

This design was implemented using Ruby on Rails which is a server-side web application framework that uses the Ruby coding language. The game itself was written in HTML, JS and CSS so it could be accessed and used in most browsers, including many cellphone browsers.

### 6.2.1 Video Game Description

In the video game, players are able to move the activities by clicking on them and dragging them to a machine and clicking again to drop them. If the player tries to place an activity on a position it cannot be placed because of any of the restrictions, this activity will be returned to its last position before being dragged and an error message will appear. An example of these restrictions could be being placed on a machine of a different type.

When an activity is placed on a machine it cannot be placed over another activity as a machine can only do one activity at a time, and the "x-axis" position it must be placed in has to be after the last "x-axis" position the previous activity from the same job has (as activities of the same job must be done in order which means that a previous activity must be completed before starting the next one). If an activity is placed by the player over another activity or before that other activity, the activity placed will be moved towards the left-most "x-axis" position it can be placed, pushing forward, if it must, the other activities from the position where it was placed. If an activity is placed after another activity, the other activity will keep its position, and the activity placed will move to the left-most possible position which would be the maximum or right-most "x-axis" position between the last position from the other activity and the position from the previous activity from the same job than the activity being placed. A visual explanation can be seen on Figure 6.5.

Another rule added for the players was that once an activity was placed, they will not be able to change its position. This was done with the intention of people making decisions in a more heuristic-like way of thinking, where decisions were known before knowing their exact results.

The size of the rectangles that represent machines is the time limit in which the activities can be done. In this case, this size was equal to the size that the time of the optimal solution represents. So, unless players gave the optimal solution for the JSSP they will not be able to place all activities correctly. The game ended when all activities are placed or when an activity placed pushed other activities or itself outside the size limit of the machine rectangles. Points were calculated relative to the number of activities placed correctly.

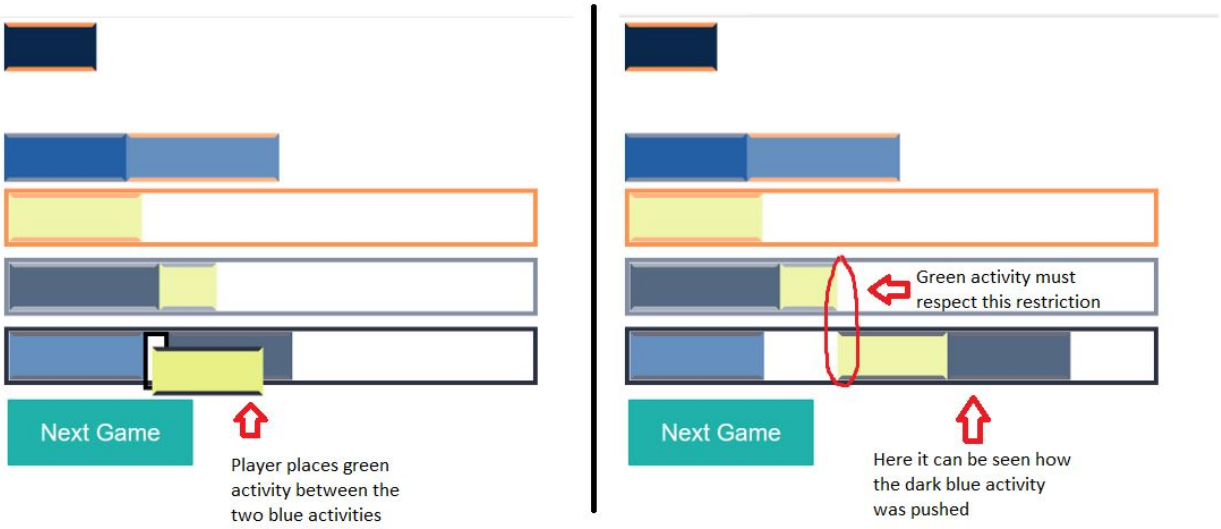


Figure 6.5: Graphical JSSP moves.

### 6.2.2 Experimentation and Results

10 JSS problems were created using Taillard's method [41], 5 of 3 Jobs and 3 Machines; 3 of 4 Jobs and 4 Machines and 2 of 4 Jobs and 5 Machines. These were the problems used in this first version of the game. Small numbers for jobs and machines were used as if not JSSPs could have become too difficult to humans to answer.

The JSS problems were given to 6 different people from different groups: 3 man of ages 13, 24 and 53; and 3 women of ages: 18, 23 and 48. These players were asked to solve the 10 JSS problems twice, finishing them all first and then giving them another try. This was a small and controlled group as there was an intention of viewing an interacting closely with each of the players so that the difficulties that these had were fully understood, also there were no resources for fully interacting the same way with a larger number of players. An interview with them revealed the following:

- All players described the game as solvable. This is fine as it would be bad if players considered the game as impossible to play because of its difficulty.
- Players were not able to come up with the right solution on many of the difficult problems (fill the machines with more than 80% of activities' time cost). This is important to consider as humans might be bad at solving JSSPs so their heuristics might also be bad (bad meaning to not be able to get a close solution to the optimal).
- Two of them when seeing too much information to process decided to place activities randomly without thinking properly on a strategy. It is good that these were not most of the players, but we can now consider that some of the humans will not use a strategy while playing.

- Five out of six described the problem as interesting and as a way they could sometime use to waste time, which means that it is good enough for using it as it is in a game.

Players also commented on how it was difficult for them to understand the game. These included viewing and understanding what the outline colors meant or trying to predict were an activity was going to end up after placing it as dependencies were not understood well. Five out of six ended up understanding the game completely after some experimentation and explaining of it. Still, with the final version of the game, players should be able to understand the rules easier and without further intervention. With this in mind, the game design was changed so it became more intuitive.

The new version of the game will also now start with low difficulty problems. JSSPs give players points relative to how good it was solved. Players will get more difficult JSSPs as they get more points (The number of machines and jobs define the difficulty of a JSSP). This was done so players can feel more comfortable with the amount of information given to them and by this trying to keep humans from using random movements to solve the JSSP because the problem seemed too large.

## 6.3 Final Version of the Video Game

The necessary changes to the interface of the game were discovered thanks to the information obtained on the first version of the video game. These included adding more visual cues. One example is that a line was added which connects the activity selected with the previous activity of the same job as seen in Figure 6.6. This way players were able to more easily recognize after which x-axis position the held activity will be placed. Another visual cue was a change in the outline color of positions where an activity can be placed. These turn yellow like in Figure 6.7 to indicate where the activity will be dropped if the player decides to release it. Colors used were also changed, but most importantly the outline color was no longer the only indicator for the machine and activity types, as now an image of an alien was used to also represent this. After players solved a JSSP, an optimal solution of it was shown to them simultaneously with their own answer with the intention of showing them their mistakes and that maybe they could improve by analyzing them (see in Figure 6.8). Points obtained by players are proportional to the amount of activities placed correctly, but also if the percentage of area from activities placed was below 85% no points are given to the user. This was done to keep players from making the bare minimum (like only placing one activity on many JSSPs) to get the amount of points needed.

The gamified graphical JSSP was transformed into a story-game by adding a narrative around the problem. This story consists on a world that the player is in charge of, where its inhabitants are able to evolve. Because of the story, players now solved the JSSPs with the intention to organize activities for the inhabitants of their planet. Solving these JSSPs earned them points which they could used in the shop to evolve the planet's population (Figure 6.9). After evolutions, players were able to see a story on how the new evolution affected their planet, making them feel in control of what happens to their world.(Figure 6.10)

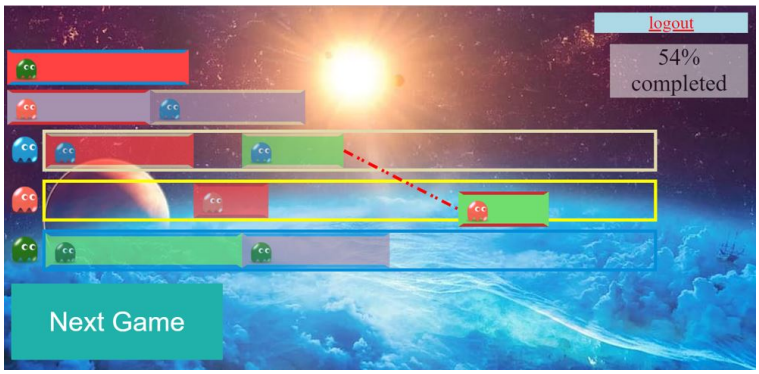


Figure 6.6: Video Game moving activities. Case 1.

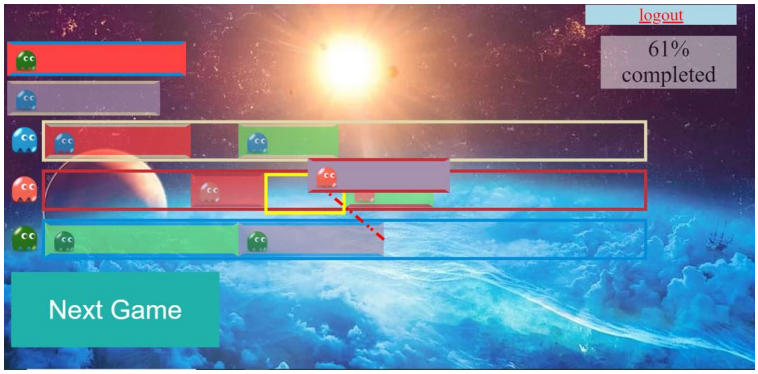


Figure 6.7: Video Game moving activities. Case 2.

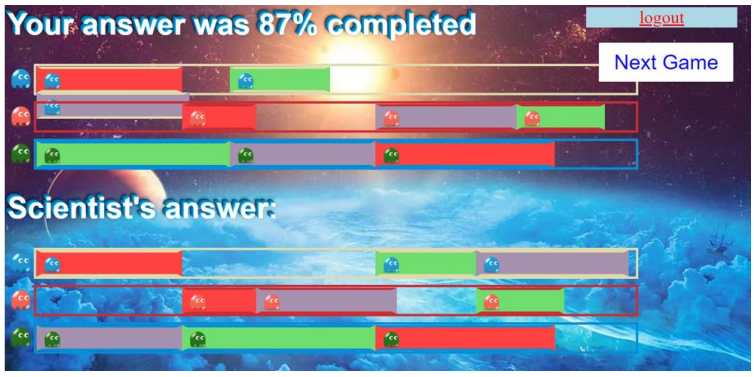


Figure 6.8: Video Game Showing Solution.

An introduction tutorial was added to the video game in which the story of the game and instructions to how to solve a JSSP in this game step by step where explained (Figures 6.11 and 6.12). With this the final version of the video game was completed as it is now able to make the players solve JSS problems, has a design and a tutorial that helps players understand how to solve a JSSP, gives them entertainment using storytelling and stores their data correctly.


Figure 6.9: Video Game Shop.


Figure 6.10: Video Game Story.

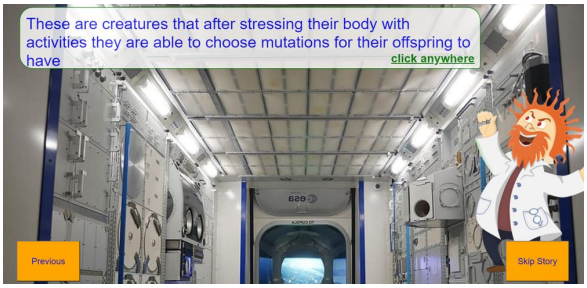


Figure 6.11: Video Game Tutorial Story.

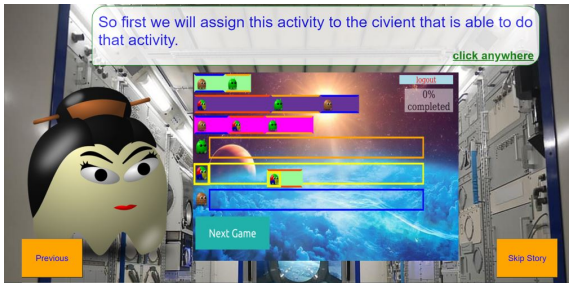


Figure 6.12: Video Game Tutorial for solving JSSP.

### 6.3.1 Analysis using Desurvire's characteristics

With the intention of creating a good game, characteristics described by Desurvire et al. [10] were considered. In this section an analysis of them in the video game is made.

- Minimize player's fatigue by varying activities and pacing during the game. In this video game there is only one main activity which is to solve the graphical JSSP, the characteristic was not done as it was considered that the game could be completed in a short time and that the stories varied enough so that it was not needed.
- Provide clear goals. It was made clear for the user in the tutorial and in the point earning system that the objective was to fill with the most activities possible the machines from the JSSP.
- Game play should have multiple ways to win. Each JSS problem had multiple ways of being solve, even if not all the ways lead to the optimal solution players still earned points with good solutions. Also, activities of the JSSP can sometimes be placed in different order but still end in the same position, so this characteristic was achieved.
- Player should not be penalized repeatedly for the same failures. There were not actual penalizations in this video game, but there were JSSP rules that could not be broken.
- Pace of the game should apply pressure but not frustrate the player. There was no time limit for solving any of the JSSPs, the player could even log out and log in later and the state of their last JSSP problem would be kept.
- Vary the difficulty so players are able to gain mastery of the game. The difficulty of the JSSPs was incremented as players gained points.
- Game should react in a consistent, challenging and exciting way to the players actions (appropriate music). It was considered that sound could be bothering for players because of the type of game, so music and sound was not added. The game did have some visual cues for the player, and the stories had some mild animation.
- Shorten learning curve by following trends set by the gaming industry. Using the mouse to drag activities was intuitive, also buttons had known symbols for their use.

- Controls should be intuitive but also customizable. Controls were simple, you could only move activities with the mouse and click the buttons with it, so there was not an option to customize them as there was no need.
- Provide immediate feedback for user actions. Many visual cues and warnings were added to help the player recognize the JSSP rules.
- The player should be able to turn on and off the game. This was done without even needing a save button, each movement players made was saved automatically and they could close the browser and login again another day and the state of the JSSP will be as they left it.

### 6.3.2 Video Game Release

Jssps used in the video game were made using Taillard's [41] method as in the first version of the game. In this previous version, the length of the machines (which translates to the maximum time to complete activities) had the optimal time for completing the JSSP as a limit. This meant that players had to place activities optimally to get all points. For this version a percentage of time was added to the optimal time to give players some space to make mistakes. JSSPs were divided by difficulty and had different characteristics because of it.

Easy levels were composed of: 20 games of 3 Jobs and 3 Machines with 1% extra time to the optimal time and 10 games of 4 Jobs and 4 Machines with 5% extra time.

Medium levels were composed of: 20 games of 4 Jobs and 4 Machines with 3% extra time, 5 levels of 3 Jobs and 3 Machines with 0% extra time and 5 levels of 4 Jobs and 5 Machines with 10% extra time.

Hard levels were composed of: 20 games of 4 Jobs and 4 Machines with 0% extra time and 10 games of 4 Jobs and 5 Machines with 5% extra time.

The game can be found in the following link https://thesis-game.herokuapp.com/, and the code for it in https://github.com/electricdrago/thesis_game. 30 players were asked to play the game, after 2 weeks, usable data from 21 players was obtained (other players ignored the request or did not play enough). With this, information from 393 games was collected.

For each JSSP a player interacts with the game saves the id of the JSSP being solved (inner id to identify it in the game), the player id and the game id. And for each movement the player makes the game stores the number of step the movement is, the id of the activity that was moved, the machine where it was placed, in which position it was placed and the time when the movement was done. When the player finishes solving the JSSP the game saves the score the player got. Admins of the video game are able to download files which have the information collected. Each file has as a name "[JSSP id]_[User id]_[Game id]_[score].csv" and contains information of the movements inside it. An example of this type of file can be seen in Table 6.1. The name of the columns in the table do not appear on the file, activity ids reference the JSSP and the machine column is for knowing in which machine the activity was placed, -1 means that it was unassigned (In the game this only happened when the user grabbed the activity but did not placed it on any machine). It is also important to know that all moves "valid and invalid" from the player are documented on the file, so it is the responsibility of the one reading the file knowing the effects of each step (invalid moves can be ignored).

Admins are also able to download JSSP information to be able to set it up and simulate players movements. An example of a file can be seen in Table 6.2. The activity Id is for a reference of the activity, but the job it belongs to and in which order it has to be completed is obtained from the job id and the position values.

| Step | Activity Id | Machine | Position | Time |
|---|---|---|---|---|
| 0 | 1411 | 2 | 0 | 2020-11-20 20:38:43 UTC |
| 1 | 1412 | -1 | 0 | 2020-11-20 20:38:45 UTC |
| 2 | 1408 | 2 | 0 | 2020-11-20 20:38:48 UTC |
| 3 | 1405 | 0 | 0 | 2020-11-20 20:38:50 UTC |
| 4 | 1409 | -1 | 0 | 2020-11-20 20:38:52 UTC |
| 5 | 1412 | -1 | 0 | 2020-11-20 20:38:54 UTC |
| 6 | 1406 | 1 | 0 | 2020-11-20 20:38:57 UTC |
| 7 | 1407 | 2 | 0 | 2020-11-20 20:39:00 UTC |
| 8 | 1410 | -1 | 0 | 2020-11-20 20:39:03 UTC |
| 9 | 1409 | 1 | 0 | 2020-11-20 20:39:06 UTC |
| 10 | 1410 | 0 | 0 | 2020-11-20 20:39:09 UTC |
| 11 | 1412 | 0 | 2 | 2020-11-20 20:39:15 UTC |
| 12 | 1413 | 1 | 2 | 2020-11-20 20:39:19 UTC |

Table 6.1: Example of game file named ”97_37_468_100.csv”.

| Act Id | Job Id | Position | Time cost | Type |
|---|---|---|---|---|
| 1405 | 0 | 0 | 60 | 0 |
| 1406 | 0 | 1 | 33 | 1 |
| 1407 | 0 | 2 | 74 | 2 |
| 1408 | 1 | 0 | 78 | 2 |
| 1409 | 1 | 1 | 40 | 1 |
| 1410 | 1 | 2 | 90 | 0 |
| 1411 | 2 | 0 | 79 | 2 |
| 1412 | 2 | 1 | 67 | 0 |
| 1413 | 2 | 2 | 97 | 1 |

Table 6.2: Example of JSSP file named ”JSSP_97_info.txt”.

## 6.4 Conclusions

The analysis in this chapter is important for future projects with similar objectives, as the focus was not only on using a game for obtaining data, but also looking on the possibility of creating or using popular games to collect data. The future planned for this project is not meant to be creating small games that can only be given to limited sets of people, but finding a way to take advantage of the growing market of video games and the large amount of data that these can generate to produce new information (heuristics in this case). Another use of transforming known problems into games is that instead of using the video games to find strategies they can also be used to solve specific problems with specific parameters directly. This should take into consideration the final size of the problem and the ability of humans to solve problems of that size.

## 6.5 Summary

This chapter explained the process that was followed when developing the video game. The first step was to define the main characteristics that it had to have. These were that it had to be of easy access, understandable and must gather the data necessary to be used in this project. After this many game designs were made, some of this, designs showed the great potential that this approach has, as problems could be placed in popular games and obtain a big amount of data from them. After this, a simple design was chosen to accommodate to the limited resources this project had. A first version of the game was made to collect data on how players interacted with the game and to be able to make improvements with the main intention of finding out if the game was understandable or not. After improving the first version of the game, a narrative was added to create a story game. This final version was distributed among some students in the campus and data which described how players solved the JSSPs was collected.

# Chapter 7

# Process for Generating Heuristics

In this Chapter human heuristics were obtained using the data collected from the video game described in Chapter 6 and the Machine Learning (ML) methods that were analyzed and selected in Chapter 5. Table 7.1 shows information about the data analyzed indicating the amount of games collected from each player. "Good Games" are all the games from the player that had a score above 80% and "Very Good Games" are the ones with a score greater than 90% ("Very Good Games" are also included in "Good Games" as these have a score above 80%). As explained in the last chapter, the score represented the percentage of activities that were placed on machines, the more activities players placed on machines the better the strategies they used for doing so should be. With the results of this chapter it was concluded that ML algorithms were capable of giving an insight of which were the heuristics used by some humans for solving the problem.

As seen in Chapter 6, the data collected from the video game only included features to replicate the players' steps and not the ones described in Chapter 5. To obtain the features selected in Chapter 5, movements in the data were simulated in their corresponding JSSP collecting the needed features. The code for this can be found in the following link https://github.com/electricdrago/thesis_JSSP_simulator.

## 7.1 Analysis of Data using ML algorithms

Similar to what was done in Chapter 5, data "Before movement" and "After movement" with activity information was also analyzed, but in this case it was not enough to predict which activity to choose, as it was also necessary to predict the position in the machine to place the activity selected. This is because, previously, heuristics only selected an activity and placed it at the end of the machine, but now, humans had the chance to select the activity and place it in any position of the machine. Because of this, machine learning algorithms were also used to choose a position on a machine, for this, "before movement" and "after movement" data sets had their corresponding machine data for training the ML algorithms to choose a position in a machine where to place the activity selected.

The process for generating data to select a machine was similar to that of selecting activities, but instead of each row representing an activity on a step, each row represents a position of a machine on which the selected activity can be placed in the state of the step. The

| Player id | Good Games | Very Good Games | Total Games |
|---|---|---|---|
| 2 | 20 | 19 | 20 |
| 3 | 19 | 13 | 20 |
| 9 | 4 | 2 | 4 |
| 11 | 6 | 3 | 6 |
| 14 | 5 | 3 | 7 |
| 20 | 18 | 12 | 21 |
| 23 | 23 | 14 | 57 |
| 26 | 17 | 12 | 20 |
| 29 | 10 | 6 | 20 |
| 32 | 8 | 4 | 12 |
| 34 | 16 | 14 | 26 |
| 36 | 4 | 4 | 8 |
| 37 | 6 | 6 | 6 |
| 38 | 14 | 8 | 40 |
| 44 | 10 | 7 | 15 |
| 47 | 26 | 18 | 35 |
| 50 | 7 | 3 | 11 |
| 51 | 4 | 4 | 5 |
| 52 | 22 | 13 | 24 |
| 61 | 18 | 12 | 19 |
| 63 | 16 | 15 | 17 |
| Total | 273 | 192 | 393 |

Table 7.1: Data from players collected in the video game (described in Chapter 6).

features used for the machines' data were: the general characteristics from Garza et al. [19] (ATP, DPT, SLACK, DNPT, NATP and NJT) which were described in section 5.4.2. and the following features (note: all features were normalized using the values from the same step):

- Will Move: It contains the amount of activities that are at the right of the position selected.
- Wasted Time from Machine: Time that the machine will spend unused if the activity is added in the selected position. Lets consider EM is the end time of the previous activity in the position where the new activity will be placed, if it would be placed in the first position the value would be 0. The value of this feature would be the max amount between 0 or the Earliest time of the activity to be placed minus EM
- Wasted Time from Activity: With the same description as the feature "Wasted Time from Machine", but the opposite, which is the max value between 0 and EM minus the earliest time from the activity.
- Total Cost: Time it takes to complete an activity normalized with the cost of all activities of the problem.
- Time Taken Machine: End time of the last activity in the machine selected.
- Position Weight: Time were the position selected is able to start (does not considers the activity selected only activities already in the machine).

- New Makespan: This feature is only considered when analyzed ”after movement”, and it is the value of the new makespan (The maximum values between the times taken by machines) of the JSSP.

### 7.1.1 First Experiments

Different clusters were created using the data obtained from the video game (data of how players solved their JSSPs) with the idea to get different heuristics from the same player. The reason why only using all games available from the player is not enough, is because humans may, and most commonly do, change their approach to the problem as they advance through the game as most of them, if not all, had never have to solve a JSS problem before. This would mean that their first games could have mainly been used for learning and experimentation, and so, the pattern/heuristic that ML algorithms are trying to recognize might not be in all the games from the player, but only in a specific portion of them.

Each group of data was analyzed using every ML algorithm selected in Chapter 5. For each group, four different sets of data were used to train the ML algorithms, one contains information from activities in the ”After movement” state, one with information from positions in machines in the ”After movement” state also, and the other two using again the information from activities and machines but now in the ”Before movement” state. The methodology was similar to what was done in Chapter 5, but instead of analyzing data from heuristics solving JSSPs, the data obtained from the video game of humans solving JSSPs was used. A sample of the accuracy from the models predicting which activity to choose can be viewed in Table 7.3. Each experiment conducted contains the same amount of data for each of the ”Before movement” and ”After movement” sets, but complete tables are not shown as the total amount of data would be too large. Instead, tables with only relevant players data and the ML algorithm that produced the highest accuracy for each player is shown, as seen in Table 7.4.

To read these tables, it is important to understand what the accuracy means. First remember that data is obtained by simulating the JSSPs and creating rows of data for each step that the player takes when solving the problem. Each row may represent either an activity or a position in a machine depending on what the ML algorithm is trying to learn. After obtaining the data, 4-split cross-validation was performed and then the average of the accuracy was obtained.

The accuracy of each model is a value which represents how many rows from the testing set did the ML algorithm labeled correctly after being trained. In this setting, however, each row represents an activity or a position in a machine

What it means is that the accuracy does not directly represent how many ”steps” from the player did the ML algorithm replicated correctly, but instead, given information of a state and an activity, the accuracy represents how many activities were labeled correctly. This is an important difference, as when it comes to evaluating activities individually from each step, there are more activities labeled as ”false” than ones labeled as ”true”. Even if the accuracy does not directly represent the steps, it is still useful to compare algorithms and players, as all had the same limitations. An advantage of having the data in this way is that values from the ML algorithms (like weights or divisions of features) are generalized for the choosing of ”true” values. This happens as in many instances the weight of special cases from the ”true” values will not have enough weight to compete with ”false” values while an ML is

trying to label an area in the hyper-plane which would make most special cases considered as ”false”. Generalizing this way can be beneficial, as humans are rational, intuitive and emotional beings, which makes it difficult for them to work in recognizable patterns for every movement they make. This would mean that in order to recognize their main behavior when solving a problem generalization is important.

As in Chapter 5, accuracy of how many ”steps” from the player can the ML model replicate correctly was also obtained. In this case the models were compared to what the players selected on all the games they played (from the specific cluster). This means that the activity and position that the player selected was compared to which activity and position did the models selected given the same state of the JSSP in which the player was (Models for choosing activity and for choosing machines tested as one, putting together the ones with the same algorithm and training set). On the next tables ”DA” will mean ”Direct Accuracy”, which would be the value obtained directly by the 4-fold cross validation for either the activity heuristic or the machine heuristic. and ”AA” stands for ”Application Accuracy” which gets the accuracy when comparing an actual selection over a group of activities. The ”Activity AA” only checks if the ML model selected the same activity as the player, but the ”Total Accuracy” (Total AA) checks if both, the activity and the position on the machine, were selected correctly. Both accuracies are relevant, the direct accuracy shows if the important features where recognize by the ML model, and the applied accuracy shows if the model is able to substitute the human in its decision making. If the direct accuracy is high but the applied accuracy is low then an analysis should be made over the model to distinguish between over-fitted values and useful ones.

Table 7.3 has the direct accuracy (DA) obtained from analyzing data of all games played by each player using the ”Before movement” state. This table is color coded, where accuracy is painted greener the larger the value is, redder the lower and yellow if it is in the middle. Accuracy values that will be shown in these tables can only be compared against other values in the same table as there is no previous work that can be compared against. Some of the models with higher accuracy will be visualized to see if these can be used to accomplish the objective of this thesis.

| | |
|---|---|
| Tree with gini and max_depth 4 | TG4 |
| Tree with entropy and max_depth 4 | TE4 |
| Tree with gini and max_depth 3. TG3 | TG3 |
| Tree with entropy and max_depth 3 | TE3 |
| Polynomial Division SVC | SVC_P |
| RBF Division SVC | SVC_RBF |
| Random Forest with gini (3 trees) | GRF3 |
| Random Forest with gini (10 trees) | GRF10 |
| Random Forest with entropy (5 trees) | ERF5 |
| Random Forest with entropy (10 trees) | ERF10 |

Table 7.2: Abbreviations from ML algorithms tested.

| Player Id | TG4 | TE4 | TG3 | TE3 | SVC_P | SVC_RBF | GRF3 | GRF10 | ERF3 | ERF10 |
|---|---|---|---|---|---|---|---|---|---|---|
| **61** | 0.764545 | 0.7203 | 0.748839 | 0.7203 | 0.726161 | 0.708969 | 0.771648 | 0.764545 | 0.763101 | 0.763109 |
| **63** | 0.71 | 0.682857 | 0.7 | 0.682857 | 0.731429 | 0.737143 | 0.724286 | 0.73 | 0.727143 | 0.737143 |
| **3** | 0.704036 | 0.679372 | 0.706278 | 0.679372 | 0.713004 | 0.711883 | 0.690583 | 0.702915 | 0.695067 | 0.707399 |
| **2** | 0.763948 | 0.734979 | 0.773605 | 0.734979 | 0.757511 | 0.771459 | 0.777897 | 0.7897 | 0.772532 | 0.795064 |
| **14** | 0.776042 | 0.6991 | 0.736979 | 0.6991 | 0.830019 | 0.799479 | 0.782907 | 0.860559 | 0.775095 | 0.784328 |
| **11** | 0.87037 | 0.898148 | 0.87963 | 0.898148 | 0.898479 | 0.935516 | 0.871362 | 0.963294 | 0.889881 | 0.962963 |
| **9** | 0.8375 | 0.705921 | 0.747368 | 0.705921 | 0.759868 | 0.747368 | 0.705921 | 0.732237 | 0.693421 | 0.758553 |
| **20** | 0.723856 | 0.738914 | 0.742259 | 0.738914 | 0.751476 | 0.755659 | 0.735559 | 0.754809 | 0.742279 | 0.757337 |
| **23** | 0.803666 | 0.779361 | 0.822188 | 0.779361 | 0.807149 | 0.813518 | 0.759673 | 0.800206 | 0.759673 | 0.797893 |
| **26** | 0.641385 | 0.66337 | 0.640042 | 0.66337 | 0.70615 | 0.696451 | 0.67993 | 0.679937 | 0.684081 | 0.67993 |
| **34** | 0.728003 | 0.682725 | 0.682701 | 0.682725 | 0.779035 | 0.781868 | 0.706841 | 0.71672 | 0.713895 | 0.71672 |
| **38** | 0.797752 | 0.723599 | 0.779779 | 0.723599 | 0.815734 | 0.793262 | 0.750606 | 0.762943 | 0.753984 | 0.75958 |
| **32** | 0.735983 | 0.628008 | 0.66001 | 0.628008 | 0.788018 | 0.740079 | 0.744112 | 0.775666 | 0.683948 | 0.759665 |
| **29** | 0.841463 | 0.813008 | 0.829268 | 0.813008 | 0.821138 | 0.821138 | 0.75813 | 0.802846 | 0.76626 | 0.808943 |
| **36** | 0.833333 | 0.742424 | 0.772727 | 0.742424 | 0.863636 | 0.878788 | 0.795455 | 0.840909 | 0.80303 | 0.856061 |
| **37** | 0.796875 | 0.578125 | 0.640625 | 0.578125 | 0.796875 | 0.796875 | 0.710938 | 0.8125 | 0.710938 | 0.796875 |
| **52** | 0.705198 | 0.700353 | 0.681089 | 0.700353 | 0.705183 | 0.698422 | 0.692646 | 0.714813 | 0.689768 | 0.711928 |
| **47** | 0.711448 | 0.723457 | 0.724656 | 0.723457 | 0.728861 | 0.733654 | 0.716865 | 0.722258 | 0.725258 | 0.721059 |
| **50** | 0.693966 | 0.685345 | 0.711207 | 0.685345 | 0.676724 | 0.693966 | 0.689655 | 0.706897 | 0.646552 | 0.676724 |
| **44** | 0.753065 | 0.720211 | 0.757181 | 0.720211 | 0.703648 | 0.724207 | 0.705731 | 0.740669 | 0.69955 | 0.72624 |
| **51** | 0.600369 | 0.573892 | 0.635468 | 0.573892 | 0.721675 | 0.721675 | 0.633621 | 0.634544 | 0.625308 | 0.677648 |

Table 7.3: ML algorithms tested with All games ”Before movement” activity information.

### 7.1.2 All Games by Player

This cluster consisted of dividing the data by player and analyzing all games from each one using the ML algorithms. Results are shown on Table 7.4. Multiple observations can be made in this table, beginning with results from player $9$ who obtained $100\%$ accuracy in the ”Activities AA” model of ”Before movement”, but the table also shows that this player only played 4 games. The model generated for player $9$ might be able to replicate the behavior that was used in the games, but as these were few, there were not enough scenarios to confirm that the player could have acted differently in other situations encountered in the JSSP.

As there is no basis to define what a ”good accuracy” means, top results obtained will be the ones considered ”good accuracy”. ”Good accuracy” values will be the ones colored as green in the tables. Other results shown on Table 7.4 are the ones from players $23$ and $61$ which had a large amount of games (57 and 19) and still obtained green numbers on accuracy. This means that the models generated for these players are more complete and have the potential to give a better insight of the players' strategy.

Player $51$ shows that a low amount of games will not result with a better accuracy. Accuracy of the models depend on the player and how recognizable are their patterns using the features defined for the ML models.

| All Games | | Before Movement | | | | |
|---|---|---|---|---|---|---|
| Player ID | Total Games | Activities DA | Machines DA | Activities AA | Total AA | ML Alg |
| 2 | 20 | 77.15% | 95.12% | 58.71% | 50.28% | SVC_RBF |
| 3 | 20 | 70.40% | 85.81% | 52.04% | 44.20% | TG4 |
| 9 | 4 | 87.04% | 82.08% | 100.00% | 81.36% | TG4 |
| 11 | 6 | 83.00% | 78.42% | 48.97% | 28.97% | SVC_P |
| 14 | 7 | 83.75% | 83.79% | 50.00% | 42.65% | TG4 |
| 20 | 21 | 73.89% | 92.74% | 40.78% | 28.43% | TE4 |
| 23 | 57 | 80.71% | 72.74% | 77.00% | 52.67% | SVC_P |
| 26 | 20 | 64.14% | 77.77% | 51.56% | 39.84% | TG4 |
| 29 | 20 | 76.29% | 73.87% | 67.39% | 53.80% | GRF10 |
| 32 | 12 | 84.15% | 83.93% | 70.94% | 54.70% | TG4 |
| 34 | 26 | 77.90% | 86.96% | 59.71% | 48.20% | SVC_P |
| 36 | 8 | 83.33% | 91.30% | 59.09% | 45.45% | TG4 |
| 37 | 6 | 79.69% | 100.00% | 71.05% | 63.16% | TG4 |
| 38 | 40 | 78.80% | 86.90% | 48.03% | 25.20% | SVC_P |
| 44 | 15 | 75.31% | 89.86% | 61.71% | 53.14% | TG4 |
| 47 | 35 | 73.37% | 87.46% | 39.66% | 31.58% | SVC_RBF |
| 50 | 11 | 69.40% | 78.11% | 52.11% | 35.21% | TG4 |
| 51 | 5 | 67.76% | 83.04% | 52.00% | 38.67% | ERF10 |
| 52 | 24 | 69.84% | 91.08% | 39.37% | 30.77% | SVC_RBF |
| 61 | 19 | 76.45% | 93.64% | 64.92% | 60.48% | TG4 |
| 63 | 17 | 71.00% | 96.98% | 52.72% | 49.79% | TG4 |
| **Average** | | 76.35% | 86.27% | 57.99% | 45.65% | |

| All Games | | After Movement | | | | |
|---|---|---|---|---|---|---|
| Player ID | Total Games | Activities DA | Machines DA | Activities AA | Total AA | ML Alg |
| 2 | 20 | 76.29% | 95.12% | 59.83% | 51.40% | SVC_RBF |
| 3 | 20 | 68.27% | 89.09% | 52.35% | 43.57% | TG4 |
| 9 | 4 | 93.52% | 90.04% | 98.31% | 86.44% | GRF10 |
| 11 | 6 | 81.51% | 85.67% | 48.97% | 32.41% | TG4 |
| 14 | 7 | 83.75% | 85.30% | 50.00% | 42.65% | TG4 |
| 20 | 21 | 73.89% | 90.22% | 43.82% | 30.96% | TG4 |
| 23 | 57 | 80.77% | 78.34% | 76.41% | 59.05% | SVC_P |
| 26 | 20 | 62.76% | 88.66% | 50.39% | 42.19% | TG4 |
| 29 | 20 | 82.13% | 78.89% | 69.02% | 56.52% | SVC_P |
| 32 | 12 | 84.15% | 88.10% | 66.67% | 58.97% | TG4 |
| 34 | 26 | 77.05% | 90.00% | 59.71% | 51.08% | SVC_RBF |
| 36 | 8 | 89.39% | 91.30% | 61.36% | 50.00% | TG4 |
| 37 | 6 | 84.38% | 91.44% | 73.68% | 65.79% | GRF10 |
| 38 | 40 | 66.80% | 94.05% | 54.33% | 29.13% | TG4 |
| 44 | 15 | 70.37% | 86.15% | 58.86% | 50.86% | GRF3 |
| 47 | 35 | 73.37% | 88.38% | 38.53% | 30.26% | SVC_RBF |
| 50 | 11 | 66.38% | 78.06% | 54.23% | 36.62% | GRF10 |
| 51 | 5 | 54.77% | 75.65% | 46.67% | 38.67% | TG4 |
| 52 | 24 | 66.09% | 88.96% | 39.37% | 30.32% | TG4 |
| 61 | 19 | 76.59% | 93.86% | 67.34% | 62.90% | TG4 |
| 63 | 17 | 71.29% | 99.25% | 52.72% | 48.54% | TG4 |
| **Average** | | 75.41% | 87.93% | 58.22% | 47.54% | |

Table 7.4: Best accuracy from all games by player.

### Good Games by Player

The second group consisted of only good games from players (Games with a score greater than 80%). This cluster was made as it could be assumed that games that had good results could contain better strategies. Some of the relevant results can be seen in Table 7.5, players that had the same amount of games in "All games" and "Good games" do not appear as these generated the same models. When comparing Table 7.5 with Table 7.4 it can be seen how players like numbers 23, 26, 29, 34, 38 and 44 got an increase in accuracy. Mainly players 23 and 38 show a great reduction of games analyzed but an improvement on their accuracy because of this. This behavior is the one expected if the player refined the strategy used while playing, but the fact that this was the behavior to look for does not prove this is true, but it could be considered a theory as for why. These results also show that this clustering did not benefited the other players, as the accuracy obtained stayed the same or got lower for them. This does not mean that clustering this way is wrong, just that it would not bring the same benefit for every player.

| Good Games | | Before Movement | | | | |
|---|---|---|---|---|---|---|
| Player ID | Total Games | Activities DA | Machines DA | Activities AA | Total AA | ML Alg |
| 3 | 19 | 67.89% | 87.41% | 48.72% | 39.74% | TG3 |
| 14 | 5 | 88.00% | 76.07% | 50.00% | 30.77% | TG4 |
| 20 | 18 | 73.80% | 90.61% | 32.89% | 23.19% | TG4 |
| 23 | 23 | 83.49% | 77.20% | 79.14% | 55.83% | SVC_RBF |
| 26 | 17 | 67.19% | 78.92% | 53.42% | 41.55% | TG4 |
| 29 | 10 | 88.02% | 90.43% | 72.86% | 60.00% | TG4 |
| 32 | 8 | 73.57% | 100.00% | 57.95% | 31.82% | TG4 |
| 34 | 16 | 73.50% | 95.36% | 65.93% | 58.24% | TG4 |
| 38 | 14 | 85.03% | 82.81% | 78.57% | 62.86% | SVC_P |
| 44 | 10 | 74.43% | 94.20% | 65.38% | 60.00% | GRF10 |
| 47 | 26 | 73.98% | 89.89% | 39.72% | 32.39% | SVC_RBF |
| 50 | 7 | 69.74% | 76.50% | 55.32% | 39.36% | TG4 |
| 52 | 22 | 68.07% | 89.84% | 43.10% | 33.74% | TG4 |
| 61 | 18 | 75.99% | 95.07% | 65.13% | 60.92% | TG4 |
| 63 | 16 | 72.73% | 94.87% | 51.50% | 48.50% | SVC_P |
| **Average** | | 77.16% | 88.04% | 59.21% | 47.45% | |

| Good Games | | After Movement | | | | |
|---|---|---|---|---|---|---|
| Player ID | Total Games | Activities DA | Machines DA | Activities AA | Total AA | ML Alg |
| 3 | 19 | 68.34% | 89.81% | 49.36% | 41.03% | TG4 |
| 14 | 5 | 89.00% | 93.99% | 50.00% | 32.31% | TG4 |
| 20 | 18 | 77.50% | 92.05% | 44.30% | 31.56% | TG4 |
| 23 | 23 | 73.71% | 90.76% | 78.83% | 64.72% | TG4 |
| 26 | 17 | 91.73% | 97.87% | 52.51% | 45.21% | TG4 |
| 29 | 10 | 79.90% | 85.49% | 72.86% | 61.43% | TG4 |
| 32 | 8 | 64.89% | 88.84% | 54.55% | 30.68% | TG4 |
| 34 | 16 | 75.87% | 98.41% | 67.03% | 57.14% | SVC_RBF |
| 38 | 14 | 85.67% | 88.28% | 82.86% | 77.14% | TG4 |
| 44 | 10 | 66.38% | 92.86% | 62.31% | 58.46% | TE4 |
| 47 | 26 | 72.98% | 90.91% | 39.01% | 31.44% | SVC_P |
| 50 | 7 | 69.08% | 72.72% | 55.32% | 40.43% | TG4 |
| 52 | 22 | 67.66% | 87.58% | 45.57% | 35.96% | TG4 |
| 61 | 18 | 74.22% | 94.37% | 67.65% | 63.45% | TG4 |
| 63 | 16 | 73.17% | 99.23% | 52.36% | 48.07% | TG4 |
| **Average** | | 77.14% | 90.90% | 77.14% | 50.01% | |

Table 7.5: Best accuracy from good games by player.

### Very Good Games by Player

With the same objective as with the group of good games, games with more than 90% were grouped by player and analyzed with the ML algorithms. It is important to consider that for some players these groupings might contain the same games as the one that had all games, while for others the amount of games could be reduced to very small amounts depending on how each player solved their problems. Important players to compare against tables from previous clusters like Table 7.5 are shown in Table 7.6 where players like numbers 47 and 52 had a slight improvement in accuracy, and numbers 3 and 26 show that even if the amount of games analyzed from them decreased the accuracy can stay the same. But the most relevant player in Table 7.6 was number 23 from which the accuracy obtained increased greatly compared to previous clusters (This means that the grouping made is having a positive effect in analyzing some players).

| Very Good Games | | Before Movement | | | | |
|---|---|---|---|---|---|---|
| Player ID | Total Games | Activities DA | Machines DA | Activities AA | Total AA | ML Alg |
| 2 | 19 | 76.73% | 93.55% | 58.19% | 50.00% | TG4 |
| 3 | 13 | 73.44% | 89.14% | 51.08% | 43.72% | SVC_P |
| 20 | 12 | 72.30% | 88.68% | 35.58% | 22.60% | TG4 |
| 23 | 14 | 74.89% | 67.75% | 85.50% | 62.00% | TG4 |
| 26 | 12 | 63.92% | 76.45% | 52.70% | 43.92% | TG4 |
| 29 | 6 | 95.08% | 94.23% | 72.50% | 57.50% | TG4 |
| 34 | 14 | 78.77% | 87.31% | 67.50% | 53.75% | SVC_P |
| 38 | 8 | 84.41% | 73.91% | 86.11% | 75.00% | TG4 |
| 44 | 7 | 75.41% | 81.87% | 62.77% | 57.45% | ERF3 |
| 47 | 18 | 69.74% | 89.79% | 48.95% | 40.56% | TG4 |
| 52 | 13 | 70.22% | 90.47% | 44.58% | 37.92% | GRF3 |
| 61 | 12 | 75.56% | 94.37% | 69.03% | 63.87% | TG4 |
| 63 | 15 | 71.15% | 95.77% | 50.00% | 46.73% | SVC_P |
| **Average** | | 77.86% | 85.58% | 60.64% | 49.48% | |

| Very Good Games | | After Movement | | | | |
|---|---|---|---|---|---|---|
| Player ID | Total Games | Activities DA | Machines DA | Activities AA | Total AA | ML Alg |
| 2 | 19 | 75.96% | 93.54% | 58.19% | 50.58% | TG4 |
| 3 | 13 | 68.56% | 92.52% | 53.68% | 47.62% | TE4 |
| 20 | 12 | 70.32% | 91.18% | 41.04% | 29.61% | TG4 |
| 23 | 14 | 82.01% | 83.24% | 83.00% | 68.50% | TG4 |
| 26 | 12 | 60.82% | 89.13% | 54.73% | 47.97% | TG4 |
| 29 | 6 | 91.80% | 100.00% | 75.00% | 62.50% | TG4 |
| 34 | 14 | 77.09% | 88.06% | 68.75% | 60.00% | SVC_P |
| 38 | 8 | 90.91% | 95.65% | 94.44% | 88.89% | TG4 |
| 44 | 7 | 65.57% | 92.50% | 64.89% | 60.64% | TG4 |
| 47 | 18 | 72.01% | 89.79% | 42.31% | 36.71% | TG4 |
| 52 | 13 | 69.88% | 91.27% | 50.00% | 43.75% | TG4 |
| 61 | 12 | 74.42% | 95.42% | 69.68% | 65.16% | TG3 |
| 63 | 15 | 69.67% | 99.16% | 54.67% | 52.34% | TG4 |
| **Average** | | 77.02% | 90.94% | 62.76% | 53.05% | |

Table 7.6: Best accuracy from best games by player.

### Last Good Games by Player

The last group, created with the notion that humans were building a strategy throughout their first games and that this strategy was not fully developed until their last games, consists on groups composed of the last 10 good games from each player. In Table 7.7 results are shown, where mainly player 3 and 34 had a good increase in accuracy when compared with Table 7.6. It might have been expected that this group would show more improvements, but it is important to remember that it will always depend on the player and if they actually used an strategy, and also that the last games from players where the most difficult.

| Last Games | | Before Movement | | | | |
|---|---|---|---|---|---|---|
| Player ID | Total Games | Activities DA | Machines DA | Activities AA | Total AA | ML Alg |
| 2 | 10 | 73.91% | 96.79% | 58.02% | 50.94% | TE4 |
| 3 | 10 | 73.84% | 84.58% | 57.36% | 51.27% | TG4 |
| 20 | 10 | 74.73% | 88.37% | 31.38% | 19.35% | TG4 |
| 23 | 10 | 73.52% | 74.69% | 68.94% | 47.20% | SVC_RBF |
| 26 | 10 | 63.70% | 74.23% | 51.95% | 43.51% | TG4 |
| 34 | 10 | 69.06% | 93.14% | 67.86% | 62.50% | SVC_P |
| 47 | 10 | 70.18% | 80.09% | 35.68% | 28.11% | TG4 |
| 52 | 10 | 70.07% | 88.24% | 33.83% | 25.87% | TG4 |
| 61 | 10 | 75.05% | 87.84% | 65.22% | 61.49% | TG4 |
| 63 | 10 | 67.77% | 94.21% | 53.42% | 51.55% | TG4 |
| **Average** | | 72.51% | 82.28% | 58.42% | 46.73% | |

| Last Games | | After Movement | | | | |
|---|---|---|---|---|---|---|
| Player ID | Total Games | Activities DA | Machines DA | Activities AA | Total AA | ML Alg |
| 2 | 10 | 77.17% | 93.57% | 58.02% | 52.36% | TG4 |
| 3 | 10 | 72.40% | 93.10% | 59.39% | 55.33% | TG4 |
| 20 | 10 | 73.97% | 91.01% | 33.14% | 22.58% | TG4 |
| 23 | 10 | 76.22% | 81.56% | 70.19% | 57.14% | TG4 |
| 26 | 10 | 65.43% | 87.61% | 45.45% | 42.21% | TG4 |
| 34 | 10 | 64.70% | 94.12% | 73.21% | 67.86% | ERF3 |
| 47 | 10 | 70.17% | 86.05% | 39.46% | 32.43% | TG3 |
| 52 | 10 | 70.07% | 90.29% | 38.31% | 30.85% | TG4 |
| 61 | 10 | 73.44% | 92.91% | 64.60% | 60.25% | TG4 |
| 63 | 10 | 68.62% | 99.22% | 49.69% | 47.83% | TG4 |
| **Average** | | 73.32% | 89.71% | 58.53% | 49.31% | |

Table 7.7: Best accuracy from the 10 last good games by player.

## 7.1.3 Portraying Heuristics

Each of the models and accuracy shown in the tables above represent an attempt of an ML to simulate a player. These models were already used to make decisions in JSSPs when obtaining the ”Total Accuracy” (TA) value. Each of the trained models can be considered as a heuristics ready to be used to solve JSSPs. But understand what the models assumed the player was doing there is a need to visualize and analyze these models. Between the ML algorithms used, decision trees are the easiest ones to visualize and understand their process. On the other hand SVMs that use kernels can get into many complications that could make it impossible to understand as features are modified with the kernel and also has a division in each of the features. The other algorithm that was use was the random forest algorithm, in this

algorithm a decision is made by multiple decision trees. Because of all of this, only decision trees will be visualized and used to understand what the player that the tree was trying to replicate was doing.

To understand results from the Decision Trees, and also any other visualization of the ML algorithms used, it is important to understand what it could mean to consider specific features to make decisions. For this, features have to be explained on what information each one gives from the activity or machine and from the state of the problem. It is important to remember that each row can represent either an activity or a position of a machine, on a state of the JSSP, and that features were normalized with rows of the same step which made them comparable with the other available activities of the state.

1. Job id: For a computer the id of the job should be negligible as it does not give relevant information on the characteristics of the job, but for players of the video game it represented position in the graphical representation of the JSSP. Jobs with the lower id were at the top, and the ones with the highest ids in the bottom. Players for whom this feature was relevant, usually completed the JSS problem job per job from top to bottom, as there was no restriction that made the players place activities in machines in order, this was not a bad strategy. Rows with a job id near 0 are activities on the top, and activities near 1 are activities at the bottom.

2. Activity Id: This variable can be used to interpret how many activities have been assigned from the activity's job compared to others. Players for whom the ML algorithms gave importance to this feature usually selected activities with an id close to 0. This meant that they solved the problem selecting the activities from which less activities have been assigned from their job.

3. Cost: Costs near to 1 are the ones with the greatest cost, and close to 0 with the lowest cost. Players could have selected the highest cost activities or the lowest cost activities and it will be seen on how ML algorithms considered this feature.

4. Time Left Job: Relevant to identify which of the jobs have more or which have less time from activities pending.

5. Earliest Time: This feature can tell which activity from the available ones can be placed first or last, but it was usually used in combination with other features to make decisions.

6. Earliest Possible Time: This feature considers the time where an activity would be placed if placed at the end of a machine of its type. Values in this feature can tell which activity from the ones available can be placed the earliest or latest.

7. Wasted Time: It helps to compare which activity when placed at the end of a machine will waste more time.

8. Time Taken by the Machines of the Activity's type / Time Taken Machine: considers which machines are the least and most filled.

9. Final time: Considers which activity will end the earliest or latest if placed at the end of its machine.

10. Will Move: Values close to 0 means that the position is close to the end of the machine, and close to one the opposite.

11. Wasted Time from Machine: Values close to 0 mean that the player tries to place the activity as earliest as possible but without wasting time from the machine.

12. Wasted Time from Activity: Values close to 0 mean that the player does cares about the ”earliest time” of the activity.

13. Position Weight: Values close to 1 means that the position is close to the end of the machine, and close to 0 the opposite.

14. New Makespan: A value close to 0 means that the machines ended up with the minimum time possible after placing the activity.

15. Average Processed Times (ATP): It gives information on what stage of the JSSP the activity is in, the lower the value will mean that it is in a beginning stage. The value can only be from 0 to 1.

16. Dispersion of Processing Time Index for Scheduled Activities (DPT): This feature gives information on the state of how different are the activities that have been assigned.

17. Percentage of Slack in Make-span (SLACK): Gives information on how much time is being wasted on machines in general, a smart objective would be to minimize this value.

18. Dispersion of Processing Time Index for Pending Activities (DNPT): This feature gives information on the state of how different are the activities that have not been assigned.

19. Average Not Processed Times (NATP): This feature is inversely correlated to ATP, so it has the same purpose but with the meaning of values reversed.

20. Average Pending Processing Time per Job (NJT): Gives information of the average size of pending jobs.

Figure 7.1 shows one of the activity models obtained from player 11. The tree is composed of nodes (squares) and edges (lines), to read a tree you start from the top-most node and if the condition inside the node is True you move to the Left and if it is False you move to the Right until you get to a leaf node. Each of the nodes has inside different information about itself, the first line is a condition, which checks the value of a feature and it determines if you need to now read the node to the Left or to the Right. The second row contains the gini or entropy value in that node, which gives you an insight of how well divided was the data that the node received while training. The next row is the number of training samples that the node received. The “Value” row shows how many samples of each class were in that node when training, and the final row shows which was the class with more samples. The two possible classes are “Choose” and “Do not choose” to decide if the activity should be the one to move next to a machine (Choose) or not (Do not choose). Let us remember that this tree is trying to replicate the behavior of player 11, so the decision it takes is the decision that it assumes player 11 would have made. Trees are color coded by how dominant is one class

over another in the training data that went through that node, this dominance can also be seen in the "value" row.

To understand what Figure 7.1 is saying about how player 11 moved, it is important to remember what the objective of the tree is, which is to decide if a single activity, on a state of the JSSP, should be the next one to assign a machine to. It is also important to remember that "descriptive features" were normalized against activities of the same state, this means that the ones with values close to 0 on a feature are the ones that had the lowest values of that feature between the activities of the same step, and values close to 1 the largest. With this information it is possible to know how a tree can be used to understand what where the rules that the player, either consciously or unconsciously, used to decide which activity to select next.

At the top of Figure 7.1, "Job ID" can be seen as the first feature to use for making a decision, there is an important thing to consider when it comes to normalization and divisions made by the ML algorithms, and this feature can be used to explain this. A common case on a step of a 3 job JSSP is to have 3 activities as available (one for each job) with the job ids of 0, 1 and 2, when normalized these values become 0, 0.5 and 1. When ML algorithms decide how to divide a feature to decide which values belong to one class or another, this is made by a theoretical line between the training sample positions over that feature. So, when receiving data from a player who chose the activities with the lowest job id available, the ML model would get in the training data many activities with the value of 0 in the job id, and labeled as True or as "Choose", and many other activities with the values of 0.5 and 1 labeled as False or "Do not choose". So, when the ML algorithm decides which value to use to make a division on the Job Id feature to decide when to label an activity as True or False, even if all the activities that were labeled as True had exactly 0 as their Job ID value, the ML algorithm will set the value of the division in a number between 0 and 0.5 which will depend on random variables, and it could be for example 0.25. This value seems to divide the data correctly, but the problem will be when the same tree is tested with JSSPs of 11 machines, where the job ids of the activities would be $0, 1, 2, 3, 4 \ldots 9, 10$ and the normalized values would be $0, 0.10, 0.20 \ldots 0.90, 1$, in this case activities with the normalized values of 0.10 and 0.20 will be selected as True even if the intention of the player was to only select activities with the lowest job id, which normalized should always be a value of 0. This is important to consider when understanding strategies of the players from reading these trees as players only solved JSSPs with a number of machines between 3 and 5.

After all these considerations, it is possible to understand the strategy that Figure 7.1 is showing. Starting from the top-most node, it can be seen that it checks if the Job ID is lower than 0.25, but as it was discussed, this probably means that activities with the lowest Job ID were selected. The next step would be to check how well divided the data ended after the first decision, for this it is only needed to see the "Value" row of the resultant nodes. In the tree of Figure 7.1 it can be seen that after the first decision classes are mostly divided, and that the 3 samples from each side that are not from the same class could be ignored, with this in mind it is possible and recommendable to ignore the rest of the tree and only use the first node to make the decision. This would mean that the strategy or "heuristic" obtained from player 11 would be to "From the available activities choose the one with the lowest Job ID".

The strategy to select an activity obtained from player 11 is to simple for it to be useful to solve JSSPs, but the complex decision on how to solve the problem can still be on the

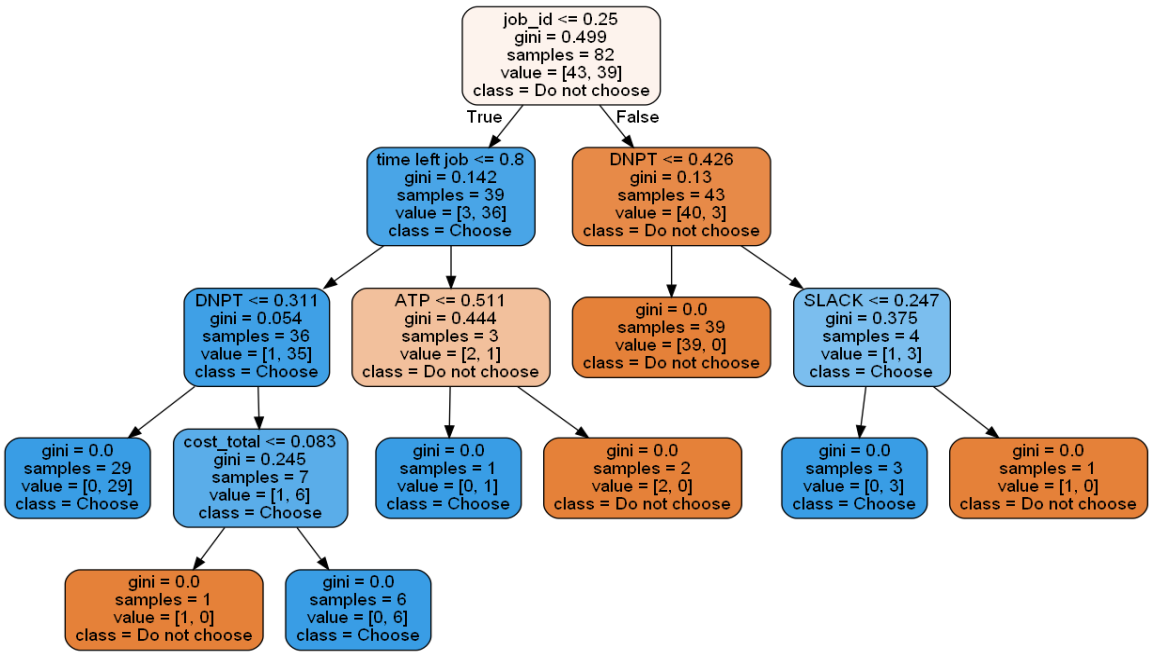


Figure 7.1: Activity heuristic generated for player 11 with information ”After movement” from the group of ”All Games”.

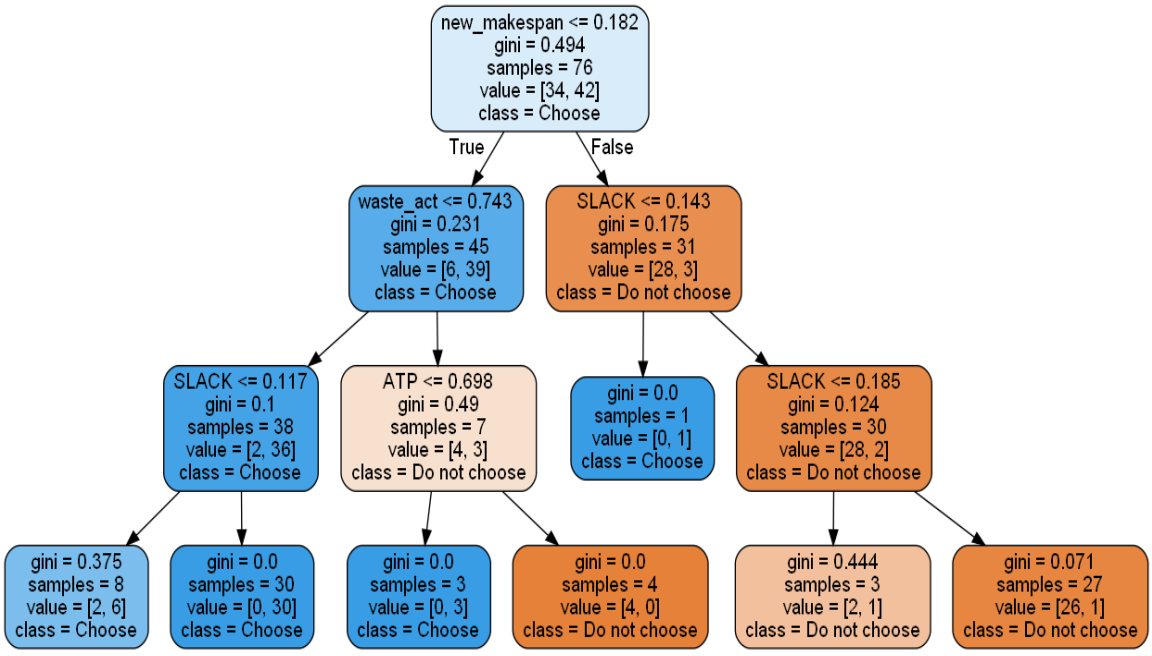


Figure 7.2: Machine heuristic generated for player 11 with information ”After movement” from the group of ”All Games”.

selection of a position in a machine. The way player 11 selected a machine can be seen in Figure 7.2. In this tree, similarly the tree in Figure 7.1, the first node can be seen as the only one relevant. In this case it classifies as “Choose” the position with the lowest new makespan, this means that the position to place the activity is the one that will end up generating the lowest makespan. And the complete heuristic for player 11 would be ”Choose the activity with the lowest job ID and place it where it would generate the lowest makespan”.

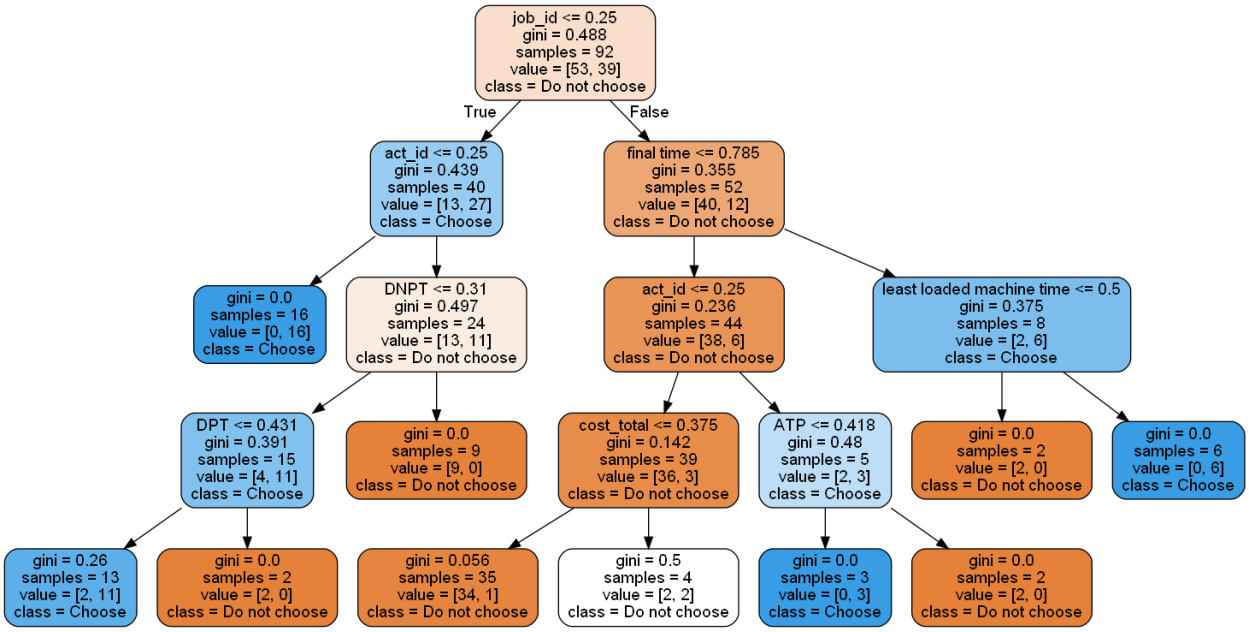


Figure 7.3: Activity heuristic generated for player 29 with information ”Before movement” from the group of ”Very Good Games”.

A more complex tree can be seen in Figure 7.3 which is the activity selection heuristic for player 29. It starts again with the job id feature, but in this case there is not a class that out-values the other in each of the nodes to the sides. At the Left the next condition tells to ”Choose” the activity if the activity ID is low, but if it is high, the next feature to check is the DNPT which is a value that represents the variety of sizes of activities that do not have a machine assigned to them, if the variety is high and the activity id is not one of the lowest, then the activity would not be selected even if the job ID was low. On the Right side of the first node, which an activity goes there when the job id is not one of the lowest, the next feature to check is “final time” which would be the end time of an activity if placed at the end of one of its machines, if this value is not high then it can be said that the activity wont be selected as the leaf node from that side that ends in “Choose” is not relevant, but if it is one of the highest, it can be said that it would be chosen. Taking into account the next node can be done even if the “Do not chose” node also seems irrelevant, just consider that least loaded machine time (which is the same as the time taken from machine) is the end time from the machine where the activity will be placed.

Player 29 also has a machine heuristic, and it can be seen in Figure 7.4. A quick interpretation of its heuristic for choosing a position in a machine made by following the nodes of

the Figure, is that it will choose a position with one of the lowest waste from machine time, but if the SLACK from the machines is not low, then it will also be checking the waste from the activity time, and if both are from the lowest possible the activity would be chosen.

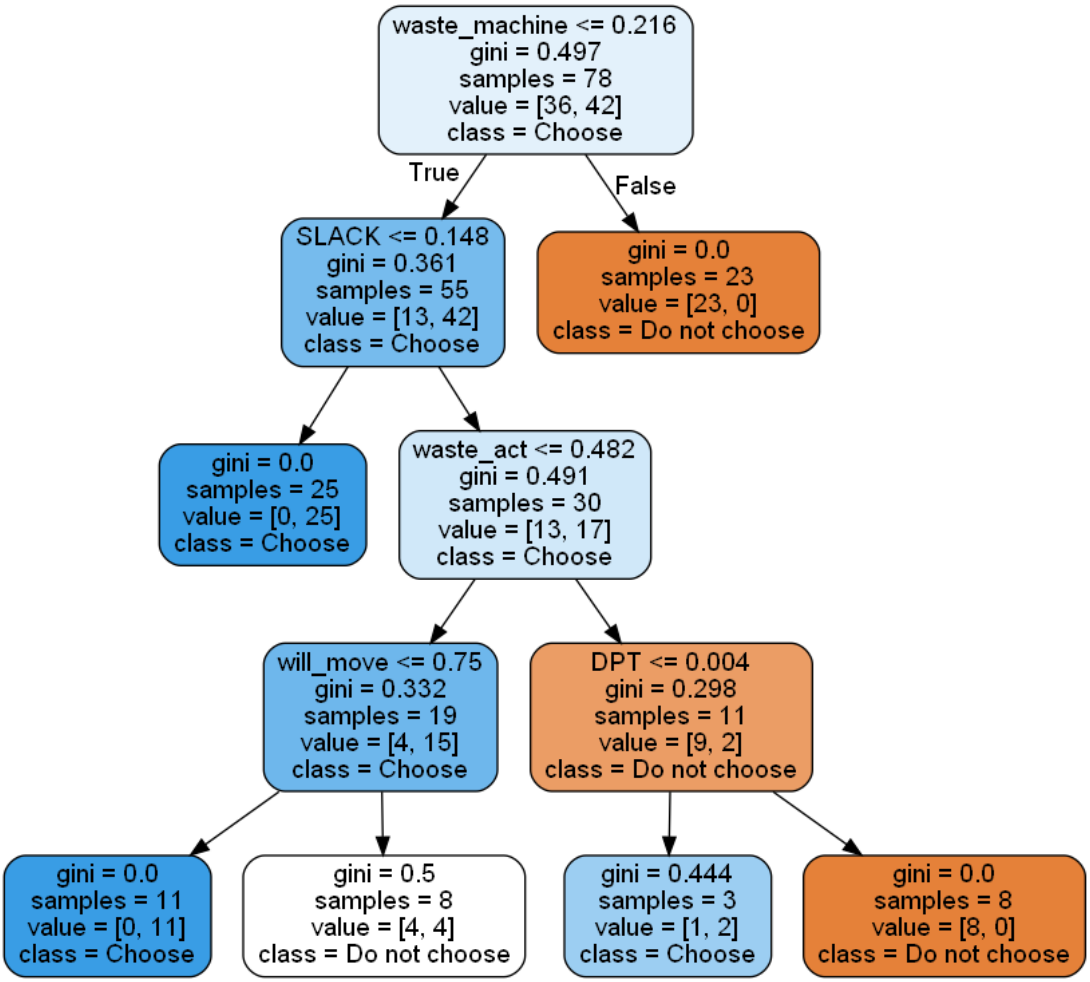


Figure 7.4: Machine heuristic generated for player 29 with information ”Before movement” from the group of ”Very Good Games”.

### 7.1.4 Comparing Human Heuristics obtained by Decision Trees with Basic Heuristics

Some of the heuristics obtained were compared against basic heuristics using Taillards [41] method (Described in Chapter 5). The values used for creating the JSSPs are shown in Table 7.8. Each of the heuristics was used to solve each of the JSSPs to compare the makespan that was obtained with each heuristic on each JSSP. The makespans obtained are shown in Table 7.9 and are color coded by column so the lowest values (which is good for makespan) appear greener and the highest ones redder. In this table there is an ”Average” row which is the average of the makespans of only the basic heuristics. This row was created as the

intention of this table is to compare basic heuristics to the ones obtained from humans, so a fair comparison would be with the average. For the comparison a column named ”Count” was placed at the end, were it shows the number of JSSPs in which each heuristics got a value which is less or equal than the average, this way heuristics could be compared. It can be seen only one human heuristic of the ones tested got a good number on the column ”count” this is because it was above the average from the basic heuristics.

Results do show that basic heuristics were superior to the human heuristics that were compared against. T-tests were done between values from basic heuristics and values of human heuristics, and values obtained do show that there is a significant difference that is not due to random chance between results obtained. Still, human heuristics showed variety, against basic heuristics and also themselves as none got the same or similar results on the makespan. This makes this thesis a success, as not only these can be considered new heuristics, but also, as the one from player 63, there seems to be a possibility of obtaining ”good” heuristics. Experiments done have a long path to go before it being used at its full potential. But for now, it proved its capacity to generate different heuristics inspired on human movements, and the possibility to generate useful heuristics.

| JSSP ID | # Jobs | # Machines | Time Seed | Machine Seed |
|---|---|---|---|---|
| 1 | 4 | 4 | 1166510396 | 164000672 |
| 2 | 3 | 3 | 840612802 | 398197754 |
| 3 | 3 | 3 | 1314640371 | 386720536 |
| 4 | 3 | 3 | 1227221349 | 316176388 |
| 5 | 4 | 4 | 533484900 | 317419073 |
| 6 | 4 | 4 | 1894307698 | 1474268163 |
| 7 | 4 | 4 | 874340513 | 509669280 |
| 8 | 5 | 5 | 1344106948 | 1868311537 |
| 9 | 5 | 5 | 425990073 | 1111853152 |
| 10 | 5 | 5 | 666128954 | 1750328066 |
| 11 | 15 | 15 | 342269428 | 1806358582 |
| 12 | 15 | 15 | 1603221416 | 1501949241 |
| 13 | 15 | 15 | 1357584978 | 1734077082 |
| 14 | 20 | 15 | 1124986343 | 1209573668 |
| 15 | 20 | 15 | 1463788335 | 529048107 |
| 16 | 20 | 15 | 1056908795 | 25321885 |
| 17 | 10 | 10 | 442723456 | 1369177184 |
| 18 | 10 | 10 | 2033800800 | 1344077538 |
| 19 | 10 | 10 | 964467313 | 1735817385 |

Table 7.8: JSSPs for comparison.

| JSSP ID | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|
| Earliest Start Time (EST) | 307 | 304 | 244 | 304 | 300 | 378 | 384 |
| Shortest Processing Time (SPT) | 489 | 288 | 464 | 331 | 526 | 470 | 475 |
| Longest Processing Time (LPT) | 498 | 258 | 487 | 527 | 568 | 593 | 577 |
| Maximum Job Remaining Time (MRT) | 337 | 327 | 279 | 304 | 494 | 389 | 505 |
| Most Loaded Machine (MLM) | 344 | 309 | 244 | 266 | 300 | 519 | 341 |
| Leas Loaded Machine (LLM) | 333 | 309 | 244 | 304 | 300 | 378 | 384 |
| Average (Avg) | 384.7 | 299.2 | 327 | 339.3 | 414.7 | 454.5 | 444.3 |
| All by player 11 after movement | 406 | 304 | 364 | 266 | 427 | 523 | 421 |
| Very Good by player 29 before movement | 501 | 257 | 487 | 492 | 534 | 614 | 437 |
| All by player 36 after movement | 664 | 345 | 487 | 376 | 663 | 620 | 609 |
| Last by player 61 before movement | 476 | 304 | 244 | 387 | 740 | 531 | 509 |
| Last by player 63 before movement | 357 | 309 | 279 | 384 | 449 | 463 | 406 |
| T-test | 0.074684 | 0.392938 | 0.26613 | 0.227684 | 0.047925 | 0.03941 | 0.2792 |

| JSSP ID | 8 | 9 | 10 | 11 | 12 | 13 |
|---|---|---|---|---|---|---|
| Earliest Start Time (EST) | 487 | 527 | 490 | 1575 | 1457 | 1528 |
| Shortest Processing Time (SPT) | 912 | 693 | 969 | 6146 | 6801 | 6642 |
| Longest Processing Time (LPT) | 1116 | 1018 | 1006 | 6401 | 7319 | 7269 |
| Maximum Job Remaining Time (MRT) | 493 | 540 | 487 | 1946 | 1725 | 1739 |
| Most Loaded Machine (MLM) | 603 | 511 | 639 | 5393 | 5664 | 4537 |
| Leas Loaded Machine (LLM) | 487 | 540 | 537 | 1678 | 1645 | 1545 |
| Average (Avg) | 683 | 638.2 | 688 | 3856.5 | 4101.8 | 3876.7 |
| All by player 11 after movement | 613 | 688 | 644 | 5685 | 5196 | 4614 |
| Very Good by player 29 before movement | 750 | 1036 | 801 | 4216 | 4699 | 4775 |
| All by player 36 after movement | 1330 | 1346 | 1337 | 9955 | 9986 | 10507 |
| Last by player 61 before movement | 699 | 868 | 543 | 7748 | 7624 | 5535 |
| Last by player 63 before movement | 677 | 671 | 613 | 3204 | 3570 | 3308 |
| T-test | 0.229549 | 0.040136 | 0.284629 | 0.082902 | 0.113673 | 0.141632 |

| JSSP ID | 14 | 15 | 16 | 17 | 18 | 19 | Count |
|---|---|---|---|---|---|---|---|
| Earliest Start Time (EST) | 1812 | 1788 | 1825 | 1001 | 1070 | 810 | 18 |
| Shortest Processing Time (SPT) | 8402 | 9415 | 9190 | 1906 | 2662 | 2650 | 2 |
| Longest Processing Time (LPT) | 9248 | 8668 | 10245 | 3055 | 3643 | 2949 | 1 |
| Maximum Job Remaining Time (MRT) | 2313 | 2335 | 2089 | 1081 | 1223 | 988 | 16 |
| Most Loaded Machine (MLM) | 7081 | 7660 | 7399 | 2150 | 2782 | 2136 | 8 |
| Leas Loaded Machine (LLM) | 1732 | 1834 | 1872 | 1081 | 1229 | 903 | 18 |
| Average (Avg) | 5098 | 5283.3 | 5436.7 | 1712.3 | 2101.5 | 1739.3 | 10.5 |
| All by player 11 after movement | 7698 | 6183 | 7276 | 2507 | 2158 | 1509 | 5 |
| Very Good by player 29 before movement | 5358 | 5658 | 5220 | 2114 | 2767 | 2289 | 3 |
| All by player 36 after movement | 12844 | 13458 | 14316 | 4113 | 4824 | 4163 | 2 |
| Last by player 61 before movement | 10619 | 9659 | 7853 | 3035 | 2859 | 3703 | 2 |
| Last by player 63 before movement | 3762 | 3903 | 4670 | 1710 | 1827 | 1637 | 14 |
| T-test | 0.104495 | 0.149198 | 0.165388 | 0.047309 | 0.137124 | 0.095726 | 0.1146 |

Table 7.9: Comparison of total makespan obtained after heuristics solved JSSPs.

## 7.2 Conclusions

In this chapter it was possible to see how accuracy obtained from each of the models varied depending on the player. In general the accuracy obtained did not seemed high, but as demonstrated with the visualization of the heuristics, the strategy used by the human players was somewhat identified by the ML model, which means that these are serving their purpose. It was also seen that clustering the data affected some of the results and it is expected that if data obtained from each player increases the clustering would have a higher effect. It is also important to notice the great advantage that decision trees have over the other algorithms in this thesis, as these are possible to visualize and understand and also got the better accuracy results, because of this it is recommended that further studies focus more on the use of decision trees to obtain results. Some of the human heuristics obtained with the decision trees were compared with the basic heuristics, the analysis concluded that in general basic heuristics were better, but some of the human heuristics are comparable to basic ones and because of this useful to use in hyper-heuristics. Overall this chapter was a success as some insight on how players played was obtained, which means that this might be a correct approach but some improvements are needed.

## 7.3 Summary

In this chapter, ML algorithms that were analyzed and selected in Chapter 5 were tested to see if these were able to obtain heuristics from human players. This was a difficult task as there is no way to know if the players actually had an intuitive, unconscious or conscious strategy while playing, or if it was used in all their games or only on a few. Because of this, many clusters for dividing data were defined to try to find heuristics on different groups of games. Data obtained from the video game was analyzed and many ML algorithms were used to train ML models. The accuracy from each model varied, but some of these were visualized to get an insight of what players thought when solving the JSSPs. It was possible to get a glance of what the players were doing while playing so their heuristics could be assumed. Some of the heuristics obtained were compared against basic heuristics and most of the human heuristics tested did not do well against the basic heuristics. This was not that relevant as the main objective of this thesis was to know if new heuristics could be obtained by analyzing data from humans solving the JSSP, which seems achievable thanks to the results.

# Chapter 8

# Conclusions and Future Work

This thesis was developed with the idea of obtaining heuristics for the JSSP by making people solve them using crowdsourcing via video games. Heuristics used in the computer science field are a bio-inspired algorithm which come from an idea in the psychology field were heuristics are one way humans try to solve complex problems. This led us to assume that humans would use heuristic when solving a JSSP. The original idea did not contemplate how the heuristics would be generated from player interaction with the video game. For this, a method was designed to be able to transform information obtained from a video game into heuristics. The approach selected was to use Machine Learning (ML) algorithms and make them learn the behavior of the players, with the expectation that the pattern recognized by the ML can be interpreted as a heuristic.

The design of the method consisted of creating features based on defined basic heuristics, generating the data and analyzing it with ML algorithms. Thanks to the features defined, some of the ML algorithms analyzed were able to almost completely imitate the behavior of basic heuristics, but these were not perfect. Because of this, there are areas of opportunity in this method. One of these is the generation of more features. This method is completely based on features, and the more the features are able to describe the state of the JSSP, the better the method would become to recognize heuristics. Another area of opportunity is the improvement of the main reason of why ML algorithms were not able to perfectly imitate heuristics even when the features for this to be possible were defined. The reason was that each activity available was considered individually (a row only contained one activity) and information contained in other activities was sometimes necessary (activities from the same steps and their information were in other rows). To avoid this, it is necessary to use methods that can analyze data in groups (a group would be the rows generated for a step). It is important to remember that being able to visualize the process used to emulate the heuristics and infer the heuristic from the parameters of the same process can be of high value as it will be easier to use good heuristics created in other problems or to understand and study them.

After the main method for analyzing data was tested, the process to develop the video game started. Many designs for the video game were considered but the one that better conformed with the resources available was chosen. The game was made and distributed among players proving to be successful on collecting data from humans solving the JSSP. An area of opportunity comes from the fact that the JSSP is complex problem, and with this level of difficulty, a longer period of interaction is required by the players to improve and develop their

strategies, as well as to achieve consistency in the results. For this reason, it is proposed as a project for future research to test the other designs (4D-Tetris and Monster Hunters), giving players the opportunity for a longer period to develop their skills and generate more valuable information. Something learned from this stage was that transforming a problem to a game is not only useful for this type of research were heuristics are obtained from the data generated, but also by representing the JSSP graphically and interacting with it, it facilitates the thinking and creation of heuristics directly from the player. With this, it is possible to experiment with different ways of approaching the JSSP or any problem, other than data analysis.

In the last section of this thesis the method proposed in Chapter 5 was used with the data collected from the video game. The intention of this chapter was to recognize the strategy used by the players for solving the JSSP using ML algorithms. As it is known humans evolve and learn it was considered that their behavior and strategy between games could change. Because of this different propositions for grouping data of their games were defined with the intention of obtaining the games were the player used the same strategy. These groups were made by basic conditions as ”last games from a player” as the amount of data obtained was not enough to test with more complex approaches. This is something where further work can be made, as the preprocessing of data and groups created with the games obtained from each player could improve to generate better emulations of humans' heuristics from the ML algorithms.

Data was analyzed in many ways and the heuristics were visualized using the decision trees generated with some of the players' data. With this, it was possible to get an insight on what type of activities players placed and where, getting with that a better description of the strategy used by them. If improvements from future work are made, like improving the method used for analyzing the data, obtaining more data from the game and upgrading the preprocessing of data, a more refined heuristic might be able to be visualized. Features used for creating machine heuristics can also be improved. In this chapter it was also concluded that decision trees are the ML algorithm with most potential, with better accuracy results and the possibility of understanding the strategy obtained, because of this it is recommended to focus more on the use of this algorithm when researching this kind of project.

This thesis has two elements that can become of high importance. The first one is the obtainment of heuristics from data analysis. This one is important as it does not have to be limited to the analysis of human data, but it can also be used to analyze data obtained differently. An idea on how to use a method able to obtain heuristics from data, would be to use it to analyze data of steps that lead to optimal solutions, which could lead to interesting results. A more specific analysis that could be made is to analyze steps of optimal solutions but from problems with specific characteristics which have a higher probability of being able to be solved with the same strategy. This could be more useful in real life as it is common that entities that need to solve problems get problems with similar characteristics. In Chapter 5 and 7 a method for doing this was proposed, and this method could be improved and polished so it is able to accomplish its objective.

The second one is in Chapter 6 were many designs for the video game were made, and with one of them it was proved that some problems can be transformed and combined into a slightly different version of a typical popular game. This is meaningful for the approach that was proposed, as using video games is not only a fresh and innovative idea to keep studying, but also a beneficial one. Video games are a growing industry in which large amount of data

can be collected. If a method like the one proposed in this thesis is successful on taking advantage of the data generated by the video game, the available data to exploit would be in the size of Big Data. This thesis could also be a next step for new projects to grow that can take advantage of one of the biggest source of data of problems being solved by intelligent beings (video games generate problems for humans, which they solved because problems are presented on an entertaining way).

The main accomplishment of this thesis was the generation of a starting point to the proposed approach, as not only a method with prospects was defined, but also many approaches and changes were proposed. There is still a very important question that has yet to be answered, which is, do humans always solve complex problems in patterns that could be recognizable? If the answer is yes, then there is a huge potential in this project and its continuation, if not, this project can still generate important models for problem solving as it faces different difficulties that can lead to the creation of innovative ideas. The main paths for future work are the enhancement of methods to obtain heuristics from data, the application and analysis of different designs that transform the JSSP into a video game, and the improvement of the preprocessing of data obtained from the video game.

# Bibliography


[1] ABRAHAM, A., BUYYA, R., AND NATH, B. Nature's heuristics for scheduling jobs on computational grids. In *The 8th IEEE international conference on advanced computing and communications (ADCOM 2000)*, pp. 45–52.

[2] AYODELE, T. O. Types of machine learning algorithms. *New advances in machine learning 3* (2010), pp. 19–48.

[3] BAKKES, S. C., SPRONCK, P. H., AND VAN LANKVELD, G. Player behavioural modelling for video games. *Entertainment Computing 3*, 3 (2012), pp. 71–79.

[4] BONACCORSO, G. *Machine learning algorithms*. Packt Publishing Ltd, (2017).

[5] BRABHAM, D. C. *Crowdsourcing*. Mit Press, (2013).

[6] BREIMAN, L. Random forests. *Machine learning 45*, 1 (2001), pp. 5–32.

[7] BRUCKER, P., JURISCH, B., AND SIEVERS, B. A branch and bound algorithm for the job-shop scheduling problem. *Discrete applied mathematics 49*, 1-3 (1994), pp. 107–127.

[8] COLORNI, A., DORIGO, M., MANJEZZO, V., AND TRUBIAN, M. Ant system for job-shop scheduling. *Belgian Journal of Operations Research 34* (1994), pp. 39–53.

[9] DE MESENTIER SILVA, F., ISAKSEN, A., TOGELIUS, J., AND NEALEN, A. Generating heuristics for novice players. In *2016 IEEE Conference on Computational Intelligence and Games (CIG)* (2016), IEEE, pp. 1–8.

[10] DESURVIRE, H., CAPLAN, M., AND TOTH, J. A. Using heuristics to evaluate the playability of games. In *CHI'04 extended abstracts on Human factors in computing systems* (2004), ACM, pp. 1509–1512.

[11] DOMINGOS, P. A few useful things to know about machine learning. *Communications of the ACM 55*, 10 (2012), pp. 78–87.

[12] EDELKAMP, S., AND LOMUSCIO, A. Automated creation of pattern database search heuristics. In *International Workshop on Model Checking and Artificial Intelligence* (2006), Springer, pp. 35–50.

[13] ESTRADA, L. E. P., GROEN, D., AND RAMIREZ-MARQUEZ, J. E. A serious video game to support decision making on refugee aid deployment policy. *Procedia Computer Science 108* (2017), pp. 205–214.

[14] FABRICATORE, C. *Gameplay and game mechanics: a key to quality in videogames*. (2007).

[15] FABRICATORE, C., NUSSBAUM, M., AND ROSAS, R. Playability in action videogames: A qualitative design model. *Human-Computer Interaction 17*, 4 (2002), pp. 311–368.

[16] FEDEROFF, M. A. *Heuristics and usability guidelines for the creation and evaluation of fun in video games*. PhD thesis, Citeseer, (2002).

[17] GALLANT, S. Extracting rules from networks. In *Neural Network Learning and Expert Systems*. The MIT Press, 03 (1993).

[18] GAMES, R. League of legends. *Riot Games 25* (2009).

[19] GARZA SANTISTEBAN, F. Feature transformations for improving the performance of selection hyper-heuristics on job shop scheduling problem. Master's thesis, Instituto Tecnológico de Estudios Superiores de Monterrey, (2019).

[20] GARZA-SANTISTEBAN, F., AMAYA, I., CRUZ-DUARTE, J., ORTIZ-BAYLISS, J. C., ÖZCAN, E., AND TERASHIMA-MARÍN, H. Exploring problem state transformations to enhance hyper-heuristics for the job-shop scheduling problem. In *2020 IEEE Congress on Evolutionary Computation (CEC)* (2020), IEEE, pp. 1–8.

[21] GARZA SANTISTEBAN, F., SANCHEZ PÁMANES, R., PUENTE RODRÍGUEZ, L. A., AMAYA, I., ORTIZ BAYLISS, J. C., CONANT-PABLOS, S., AND TERASHIMA MARÍN, H. A simulated annealing hyper-heuristic for job shop scheduling problems. In *2019 IEEE Congress on Evolutionary Computation (CEC)* (2019), IEEE, pp. 57–64.

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

[23] GIGERENZER, G. *Heuristics*. Mit Press, (2006).

[24] GILOVICH, T., GRIFFIN, D., AND KAHNEMAN, D. *Heuristics and biases: The psychology of intuitive judgment*. Cambridge university press, (2002).

[25] GOOD, B. M., AND SU, A. I. Games with a scientific purpose. *Genome biology 12*, 12 (2011), pp. 135.

[26] JOHNS, M. B., MAHMOUD, H. A., WALKER, D. J., ROSS, N. D., KEEDWELL, E. C., AND SAVIC, D. A. Augmented evolutionary intelligence: combining human and evolutionary design for water distribution network optimisation. In *Proceedings of the Genetic and Evolutionary Computation Conference* (2019), pp. 1214–1222.

[27] LENAT, D. B. Eurisko: a program that learns new heuristics and domain concepts: the nature of heuristics iii: program design and results. *Artificial intelligence 21*, 1-2 (1983), pp. 61–98.

[28] LORIA, E., AND MARCONI, A. Exploiting limited players' behavioral data to predict churn in gamification. *Electronic Commerce Research and Applications 47* (2021).

[29] MAGEE, J. F. *Decision trees for decision making*. Harvard Business Review, (1964).

[30] MARION, B. Turing machines and computational complexity. *The American Mathematical Monthly 101*, 1 (1994), pp. 61–65.

[31] MAVANDADI, S., FENG, S., YU, F., DIMITROV, S., YU, R., AND OZCAN, A. Biogames: a platform for crowd-sourced biomedical image analysis and telediagnosis. *GAMES FOR HEALTH: Research, Development, and Clinical Applications 1*, 5 (2012), pp. 373–376.

[32] MORSCHHEUSER, B., HAMARI, J., AND KOIVISTO, J. Gamification in crowdsourcing: a review. In *2016 49th Hawaii International Conference on System Sciences (HICSS)* (2016), IEEE, pp. 4375–4384.

[33] PFAU, J., LIAPIS, A., VOLKMAR, G., YANNAKAKIS, G. N., AND MALAKA, R. Dungeons & replicants: automated game balancing via deep player behavior modeling. In *2020 IEEE Conference on Games (CoG)* (2020), IEEE, pp. 431–438.

[34] PFAU, J., SMEDDINCK, J. D., AND MALAKA, R. Towards deep player behavior models in mmorpgs. In *Proceedings of the 2018 Annual Symposium on Computer-Human Interaction in Play* (2018), pp. 381–392.

[35] PINEDO, M., AND HADAVI, K. Scheduling: theory, algorithms and systems development. In *Operations Research Proceedings*. Springer, (1991), pp. 35–42.

[36] ROMANOV, D., AND HOLLER, S. District heating systems modeling: A gamification approach. *Energy Reports 7* (2021), 491–498.

[37] ROSS, N., KEEDWELL, E., AND SAVIC, D. Human-derived heuristic enhancement of an evolutionary algorithm for the 2d bin-packing problem. In *International Conference on Parallel Problem Solving from Nature* (2020), Springer, pp. 413–427.

[38] SILVA-GÁLVEZ, A., MONROY, R., RAMIREZ-MARQUEZ, J. E., AND ZHANG, C. A video game-crowdsourcing approach to discover a player's strategy for problem solution to housing development. *IEEE Access 9: 10.1109/ACCESS.2021.3103930* (2021).

[39] SUTTON, O. Introduction to k nearest neighbour classification and condensed nearest neighbour data reduction. *University lectures, University of Leicester* (2012), 1–10.

[40] SYARIF, A., PAMUNGKAS, A., KUMAR, R., AND GEN, M. Performance evaluation of various heuristic algorithms to solve job shop scheduling problem (jssp) (2021).

[41] TAILLARD, E. Benchmarks for basic scheduling problems. *european journal of operational research 64*, 2 (1993), pp. 278–285.

[42] VAN EKERIS, T., MEYES, R., AND MEISEN, T. Discovering heuristics and metaheuristics for job shop scheduling from scratch via deep reinforcement learning. *ESSN: 2701-6277* (2021).

[43] VANNELLA, D., JURGENS, D., SCARFINI, D., TOSCANI, D., AND NAVIGLI, R. Validating and extending semantic knowledge bases using video games with a purpose. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)* (2014), vol. 1, pp. 1294–1304.

[44] WITTEN, I. H., FRANK, E., HALL, M. A., AND PAL, C. J. Practical machine learning tools and techniques. *Morgan Kaufmann* (2005), pp. 578.