International Journal of Computer Information Systems and Industrial Management Applications.
ISSN 2150-7988 Volume 12 (2020) pp. 231-238
© MIR Labs, www.mirlabs.net/ijcisim/index.html

Received: 15 April, 2020; Accepted: 21 May, 2020; Published: 20 June, 2020

Reinforcement Learning Environment for Job Shop Scheduling Problems

Bruno Cunha[1], Ana Madureira[2], and Benjamim Fonseca[3]

[1,2] Interdisciplinary Studies Research Center, Institute of Engineering - Polytechnic of Porto,
Porto, Portugal
bmaca@isep.ipp.pt
amd@isep.ipp.pt

[3] INESC TEC and University of Trás-os-Montes and Alto Douro (UTAD),
Vila Real, Portugal
benjaf@utad.pt

Abstract: **The industrial growth of the last decades created a need for intelligent and autonomous systems that can propose solutions to scheduling problems efficiently. The job shop scheduling problem (JSSP) is the most common formulation of these real-world scheduling problems and can be found in complex fields, such as transportation or industrial assemblies, where the ability to quickly adapt to unforeseen events is critical. Using the Markov decision process mathematical framework, this paper details a formulation of the JSSP as a reinforcement learning (RL) problem. The formulation is part of a proposal of a novel environment where RL agents can interact with JSSPs that is detailed on this paper, including a comprehensive explanation of the design process, the decisions that were made and the key lessons learnt. Considering the need for better scheduling approaches on modern manufacturing environments, the limitations that current techniques have and the major breakthroughs that are being made on the field of machine learning, the environment proposed on this paper intends to be a major contribution to the JSSP landscape, enabling academics from different areas to focus on the development of new algorithms and effortlessly test them on academic and real-world benchmarks.**

Keywords: Reinforcement Learning, Job Shop Scheduling, Simulation, Optimization, Machine Learning.

I. Introduction

The industrial growth of the last decades stimulated the necessity for intelligent systems to efficiently support manufacturing environments, since they must have the ability to rapidly adjust to unforeseen events. The scheduling procedure of modern, real-world manufacturing environments still has many difficulties in dealing with unforeseen events, and the human decisions that are made do not convey into optimized plans. Hence, intelligent and autonomous systems are required so that automatic solutions to scheduling problems can be found quickly and as optimized as possible. Even a small reduction of the time required to calculate an improved schedule can have a significant impact in the performance of an industrial business.
Scheduling has always been one of the most complex and impacting problems that the scientific community has researched, with scientists from artificial intelligence, operational research and scheduling theory [1] combining to propose several methods that attempt to solve scheduling problems. JSS is the most used formulation of the scheduling problem and is commonly applied in fields such as transportation (e.g. flight scheduling, staff allocation) or industrial assemblies (e.g. task distribution, resource assignment). A Job Shop is a setting where certain resources exist and must execute specific operations. Solutions to a JSSP must define where each operation will be executed and at which time interval that will happen. This is a classical optimization problem, where the difficulty is in finding the schedule that best takes advantages of the resource's usage.

The theory that humans learn by interacting with the environment is, probably, the most natural one and the easier to accept when we consider the nature of learning [2]. This is the key concept behind RL: perform an action, understand its effect, and learn something from it. This cause and effect relation is sustained by the scientific investigation of the human behavior conducted by psychologists [3].

The last decade has seen a huge expansion of the machine learning field. The computational power, which was previously a bottleneck for machine learning researchers, is now so widely accessible and incredible powerful that breakthroughs are being made constantly. RL, a subfield of machine learning, has benefited greatly not only from these increases of computation power, but also from the recent innovations on how to train agents to solve a specific problem. Taking that into consideration, the main contributions of this paper are the formulation of JSSPs as a RL problem and a proposal of an original RL environment that shall allow the application of the latest RL techniques on JSSPs. Considering the available options [4], RL emerges as the clear choice to solve many of the limitations of the current methods that are used to solve JSSPs.

To use RL on JSSPs two key components must be developed: an intelligent agent that makes the scheduling decisions and an environment where that agent learns how to act. This paper focuses on the environment component. Although one is not very useful without the other, the correct implementation and execution of an environment is crucial: an inferior agent could still be capable of achieving a solution, but a mediocre environment might make it so that all solutions are invalid (e.g. if the rules are not validated correctly). Also, there is only need for one environment; after its development, several agents with specific learning algorithms can then be made to achieve the best possible results.

Considering the need for improved scheduling procedures on modern manufacturing settings, the limitations that current techniques have and the major breakthroughs that are being made on the field of machine learning, the environment proposed on this paper aspires to be a major contribution to the JSSP landscape.

The remaining sections are organized as follows: Section II starts by presenting an overview of the key concepts of the related work, fundamental to the proposal that this paper puts forward; Section III proposes a formulation of the JSSP as a RL problem; Section IV contains the details of the developed JSS environment for RL agents; and, at last, section IV contains the final conclusions and puts forwards the planned future works.

II. Literature Review

This section contains a summary on important topics that are related to the work proposed in this paper.

A. *Job Shop Scheduling*

The Job Shop problem is a scheduling problem that consists on the allocation of *n* manufacturing orders (known as jobs), J_1, J_2, J_3, … J_n, in *m* machines, M_1, M_2, M_3, … M_n, which are physically available in an industrial environment, such as a factory or a workshop (hence the job shop name). Each manufacturing order is characterized by a specific number of operations. Each of these operations is represented by o_{ij}, where *i* represents the order that the operation belongs to and *j* represents the precedence of the operation (e.g. o_{23} symbolizes the third operation of the second order).

Hence, a Job Shop problem P is defined by the collection of machines, orders and operations, which establishes P as the (M,J,O) set.

Each operation also as an associated processing time, p_{ij}, that is known, which represents the number of time units that is necessary to completely process operation o_{ij}.

On a job shop environment there are some basic restrictions that must be respected. These are:

- All operations of a job can only be executed when the previous operation is completed, except in the case of the first one.
- The operations of a given job cannot take precedence over operations of another job, i.e. there is only precedence between operations of the same job.
- An operation that has already started (i.e. is being processed) cannot be interrupted.
- A machine can only execute one job at a time.
- A job can only be executed by one machine simultaneously, i.e. machine changes can only be made between operations.

Even that these restrictions may seem like a simplification of the problem that is faced by several real-world industries this approach is still very useful and advantageous, given that it allows us to obtain very valuable information, e.g. the best order of execution of the available jobs [5].

Calculating a scheduling plan is, after all, a relatively simple task. Considering that we have N jobs that must be processed on M machines at specific times, the complexity in JSSPs is the calculation of the starting time of each operation on its respective machines. However, a scheduling plan, by itself, is of little use. What is required, naturally, is an optimized dispatching plan that presents the optimum solution. Unfortunately, there is no viable way to calculate an optimized plan given the complexity of this problem.

The JSSP is considered to be of very hard resolution, and is classified as NP-hard [6]. Actually, of all the NP-Hard problems, JSSP are one of the hardest considering how complicated they are to solve [7].

From a large set of possible plans, it is necessary to choose the one that offers the best performance (according to the metric chosen). The possible combinations are quite high: considering *n* orders on *m* machines, the number of possible plans will be $n!^m$. A small problem of 5 machines with 5 orders originates 24883200000 possible combinations; and a (still relatively small) problem of 10 machines and 10 orders creates a troublesome combination amount: $10!^{10} = 3959^{65}$. A great part of these solutions would be invalid (due to violations of the basic restrictions) but, nevertheless, it is unfeasible to validate such a high number of solutions.

Considering its complexity, there are no algorithms to obtain optimal solutions to large JSSP in a timely manner. It is therefore necessary to use approximate approaches to solve this problem, allowing to find solutions that are not optimal, but are good enough to put into production.

When the optimization of something is mentioned this usually refers to the choice of an option (from a diverse range) that will minimize or maximize a certain objective function. The most common objective when solving a JSSP is to create a solution that minimizes the makespan [4] – the time at which the last operation is concluded. Even small improvements to the makespan of a plan could have a great impact on the costs and efficiency of a manufacturing system [8].

JSSPs have always been a focus of the scientific community, especially in the disciplines of artificial intelligence and operational research [9]. The first publication that analyzed the performance of algorithms solving the JSSP was made by R. Graham, in 1969 [10]; But even today it is still a very active field, since the complexity of the problem keeps attracting researchers to it.

Considering a recently published literature review, the general consensus methods to solve a JSSP, nowadays, is to use a metaheuristic [4]. The advantages are the speed at which it can achieve a good solution and the low computational effort.

However, there is still much room for improvements [11]. The solutions obtained by metaheuristics are, mostly, just good enough to be used but far from the optimal ones; they tend to not generalize well to instances of other problems (it might be successful on a specific JSSP instance but fail completely in another one); and a significant effort is required to develop an effective metaheuristic solver [4], [12].

B. Reinforcement Learning

Machine learning can be divided in three fields [13]: supervised learning, unsupervised learning and RL. Supervised learning problems operate with a labelled dataset, i.e. we know precisely how to classify each example in the data. Unsupervised learning also uses a traditional dataset, but it is unlabeled, i.e. we do not know the correct classification of each data point and have no information on the relations between examples. RL is remarkably different from its counterparts given that it uses no pre-existing data.

Essentially, RL can be defined by the process of an agent learning the best actions based on feedback provided by the environment [4]. The feedback contains a reward for the agent, which interprets it to draw a conclusion of the effects of the chosen action. This cause and effect (usually labelled as action and reward) relation is inspired in the scientific research of the human behavior conducted in psychology, going back as far as the beginning of the twentieth century [14], [15].

Any problem of reinforcement learning has two main components: the agent and the environment. The agent, which needs to have a well-defined goal, is the entity that decides the actions to be taken and that can ascertain the state of the environment, even with uncertainties. The environment is where the agent operates and is related to the problem to be solved (e.g. in a chess game, the environment will be the board). However, besides the agent and the environment there are four critical components to any reinforcement learning system: the reward, the policy, the value function of each state and the environment model [4].

The policy is what defines the behavior of the agent. The policy maps the states of the environment to the actions that the agent must have in those states. The manner in which it is defined can be simple (a table with the mappings) or quite complex (intelligent search methods), with options being stochastic and having an associated probability [16]. Thus, the policy is a core component of a reinforcement learning system, as it is sufficient by itself to establish what behavior the agent will have.

The reward is how the agent's goal is defined. After each agent action, the environment returns the reward. The goal of an agent is to maximize the total reward received throughout its interaction with the environment, regardless of the type of problem. Thus, and drawing a parallel with Thorndike's effect law [14], the reward has a major effect on the iterative construction of the agent policy and establishes what actions the agent should take: if an action chosen by the current policy receives a low reward, the policy should be updated to choose another available action when the agent is in a similar situation again.

If the reward is related to immediate feedback from the environment, the value function is what allows the agent to take a long-term view. The value of a state is the total reward that an agent can get from that state, i.e. the value indicates how positive a state is considering future states and the reward they may give. Without rewards there could be no state value, since the sole purpose of the value is to estimate how a greater total reward can be obtained. However, the value of a state is more important when the agent has to consider all available actions. The agent should opt for actions that lead to states with the highest value and not the highest reward, because then they will accumulate a higher total reward in the long run. As would be expected, it is much more difficult to determine a correct value function than a reward. While the rewards are given directly by the environment, the value should be estimated continuously through the agent's interactions with the environment. This component is, in the author's opinion (and according to one of the fathers of modern reinforcement learning, Richard Sutton [17]) the most important of any system implementing algorithms of this problem category.

The environment model is the component that seeks to replicate the behavior of the environment, so that inferences can be made and predict how it will react to an action. Given a state and an action, the model should return the reward associated to that action and calculate what the next state of the environment will be. It is the model, through inferences, that allows decisions to be made about the action to be taken before performing this action, based on expectations of what will happen in future situations. There are simpler reinforcement learning algorithms that do not use models, working simply on the basis of cycles of trial-error repetitions; by contrast, the approaches that use models are more robust, being able to make those decisions and plan their actions with a long-term view.

The data for a RL problem is generated dynamically. The environment is responsible for providing the feedback on the action that the agent performed. With that feedback, the agent will update its beliefs on what are the best actions are. Then, it will act and learn from feedback again, repeating this process over and over; effectively, this is the training loop of a RL algorithm, and where the learning occurs. The RL algorithm that controls the agent guides its discovery of what the best decisions are in order to maximize a reward; i.e. the agent is not told what to do, but instead must realize which actions provide the best rewards by attempting them (hence the cause effect paradigm).

Given that there is no need for pre-existing data and the extraordinary results that were achieved recently (e.g. DeepMind’s AlphaGo [18] or the demonstration of ambidextrous manipulation skills [19]), it is unquestionable that RL is one of the most powerful and promising fields of research, envisioned by many as the future of artificial intelligence [20].

However, RL applications are not yet unbeatable. Currently, the main shortcomings of RL are the enormous amounts of time required to train an agent and the difficulty in creating an agent that can act skillfully on environments that require reasoning or memory. The time and energy required to train agents is expected to go down as the field grows, since optimized versions of existing algorithms appear at a good rate, and novel methods in development are more focused on the runtime. The lack of capacity to reason or memorize may be an issue, but promising approaches have demonstrated how to teach agents using natural language instructions and small demonstrations with success [21], [22].

C. Environments, Actions and Rewards

A RL environment is the world were the agent is operating, which reacts to its actions. From the agent's perspective, there is a goal (e.g. in a chess game, the goal is to win it) and it is necessary to interact with its current environment to achieve it. The actions of the agent affect the environment and, in doing so, change the options that are available in future interactions.

From the environment perspective, it always is in one of many possible states. Whenever an action is performed by an agent (e.g. moving a piece in a chess game), the environment takes it as input and adapts its internal configuration in response, thereby changing to another state.

Except in the utopian case of the existence of a perfect information environment, the effects that an agent's action may cause cannot be fully predicted [23]. Therefore, the agent must monitor the environment constantly. However, the agent always knows what its goal is, and can monitor the progress that is being made (e.g. an agent that plays chess knows if it is closer to winning).

The agent makes use of the knowledge gathered by performing several actions and changes its beliefs, improving its ability to achieve the proposed goal (e.g. a chess-playing agent will be better at deciding which move will improve its chances of winning the game). The knowledge that the agent has is, essentially, a result of its exploration of the environment, through the validation of its goals (i.e. if it is achieved) and the rewards that are given by the environment (in response to the actions).

The beliefs of the agent express how to act on a state. Pragmatically, it is a mapping from all the known states of the environment to the action to choose whenever that specific state is the current one; it is a parallel with what psychology identifies as sets of stimulus-response associations [2]. These beliefs are known as the agent's policy, and this is the core of the agent since it can predict its behavior.

The reward that is provided as feedback to the agent classifies actions as good, neutral, or bad. Consequently, it is essential to the definition of the goal of an agent: to maximize the rewards that it collects from the environment over time; e.g. if an action provides a low reward, the agent updates its beliefs (i.e. its policy) so that it chooses another action the next time it faces a similar situation.

Rewards are immediate, received as feedback of actions. But it is possible to think of actions that provide lower rewards that may lead, at some point, to greater rewards (e.g. sacrificing a rook to conquer a queen in a chess game). That is the purpose of the internal value function of the agent: it tries to predict the total reward an agent might accumulate from the current state. In other words, it attempts to forecast the long-term reward of moving to a specific state, considering the states that are expected to emerge afterwards and the future rewards that they will provide.

Considering the aforementioned information, the argument can be made that a high-quality environment is essential to have any success when dealing with a RL problem. Even in a situation where the finest, state-of-the-art RL algorithms are being used, it will all be worthless if the environment in question is not capable of providing good feedback.

III. JSSP Formulation

This section explains the characteristics that make the JSSP so interesting and unique, what should be done so that it can be solved using RL techniques, and how are RL and JSSPs defined.

A JSSP consists in N jobs that must be processed on M machines at specific times, as detailed on section II.A, with the most common goal being the calculation of a solution that minimizes the makespan [4]. The conditions to define a RL problem are the specification of a goal to achieve, a set of possible actions, the states that the environment can be in and the policy of the agent. To define this problem, and as commonly used on solutions to RL applications [2], we propose the formulation of the JSSP as a Markov decision process (MDP).

The MDP is composed by two elements: the entity that makes the decisions and the environment. The entity observes the state of the environment and chooses which action to take. That action is then executed on the environment, which will cause it to change to another state. As a result of the execution, the environment presents a reward to the entity that is making the decisions. With that reward, the entity's objective is to discover the best way of making a decision in order to maximize the rewards that it gets. To reach the goal, the agent should be capable of interacting with a JSS environment where the actions that are available must allow it to allocate each job to a specific machine.

The mathematical formulation of an MDP consists in a set of finite states S (with $s \in S$), a set of possible actions, A(s) in each state, a reward function, R(s) and a transition probability between a specific state and the current state, knowing that action a was chosen. The transition probability is formulated in equation (1), representing the probability of transitioning to state s' from state s if action a is taken.

$$P\,(s', s \mid a) \tag{1}$$

Most real-world environments make it impossible to know the transition probabilities between all states, and that is where the learning mechanisms of the RL agent that is solving the problem operate. As such, the formulation of a JSS MDP requires the definition of the states (S), actions (A) and a reward function (R).

At first glance, the set of actions A of a JSSP seems reasonably straightforward to define. At any moment there is only one type of action that can be done: assign an operation to a machine. However, having only one type of action does not mean that the problem is not complex. Deciding which operation will be allocated next is the core of a scheduling problem. If we consider a problem with 50 jobs, each with 10 operations to be executed on 10 machines, and assuming that no jobs are planned into the future, at any moment, the number of operations that can be allocated will be 500. Hence, the action space A shall contain all the possible allocations that can be made at a specific moment, and the key decision that the RL agent shall take is which of the allocations shall be done.

The set of states' S contains all possible states of a RL problem. For a JSSP to be converted into an MDP, it is necessary to deliberate that the state only changes whenever a new job allocation is made. Considering that all states are not subjects of the previous states (i.e. in equation (1), *s'* is only conditioned by *s* and a), set *S* is in accordance with the MDP requirements [24]. This paper proposes that *S* is the collection of all solutions resulting from the possible permutations that a scheduling plan may have. This means that there is a state that represents each possibility of a job allocation to a machine. However, it is important to state that this does not mean that the implementation of the environment must make this calculation beforehand (or ever, for what it is worth). Calculating such a huge amount of possibilities would make this environment very slow on anything but machines with high computational powers; and that is a deal breaker for RL algorithms, since millions of iterations must be made in order to learn. What this definition means is that the set of states *S*, formally, contains all those permutations. In practice, the learning algorithm has to decide how to handle this information; e.g. a feasible approach would be to disregard all the possible permutations, focusing instead on the information of the state that is being experienced and understanding the effect that the chosen action will have (leading to *s'*).

The reward function determines the expected reward that is received after transitioning from one state to another. To define a reward, it is necessary to consider what the goal of the agent is. This proposal suggests that the JSS formulation (as a RL problem) shall use the makespan minimization as its objective. The reward shall be a ratio that estimates the benefit of the chosen action against its cost, considering the makespan minimization goal. The proposed formula for the reward function R is presented in Figure 1. The makespan difference is calculated using the known optimum values for the problems that are being solved.

$$R = \begin{cases} \textit{Invalid operation allocation}, & -1.0 \\ \textit{Valid operation allocation}, & 0.0 \\ \textit{Complete allocation and makespan diff} < 250, & 1.0 \\ \textit{Complete allocation and makespan diff} < 450, & 0.6 \\ \textit{Complete allocation and makespan diff} < 650, & 0.4 \\ \textit{Complete allocation and makespan diff} < 1500, & 0.1 \\ \textit{Complete allocation and makespan diff} > 1500, & 0.0 \end{cases}$$

Figure 1. Recommended reward function

Considering the way that RL approaches work and the nature of JSSPs, the order that the actions are taken can have a major impact on the total reward. The agent shall learn how to best navigate the decision paths that can be taken, discovering how to achieve its goals.

With the proposed formulation of the JSSP as a MDP, an agent with learning capabilities would be able to evaluate its environment, analyze its state and choose the actions that will maximize the total reward that is collected, taking into account its predefined goal.

IV. JSS Environment For RL Agents

Recent developments in RL have achieved incredible results (see section II.B). Analyzing the success that DeepMind's AlphaGo [25] achieved, it is clear that it was propelled by the newfound capacity of its novel algorithm to learn from millions of simulated games of Go. But to learn from that huge amount of games, an accurate simulator of the game of Go had to be developed (amongst many other components). AlphaGo - the agent that was created – used an appropriate simulator of the game of Go to play over and over against itself. Without the environment that simulated this game, even these advanced algorithms would not be capable of achieving satisfactory results.

Simulated environments that allow RL agents to effortlessly interact (and, hence, learn) are the key to the recent breakthroughs in RL. Nevertheless, these innovations are very recent, and no work has been done yet in the scheduling optimization field and, precisely, on the construction of a scheduling environment to train and evaluate agents that aim to solve the JSSP. That is the main motivation for this work; the design and implementation of a RL environment for the proposed JSSP.

In 2016, OpenAI released OpenAI Gym [26], which proposed an unified standard for RL benchmark problems and how the environments are implemented. This standard provides two major advantages: the ability to create comparisons between RL algorithms and the fact that it enables academics to specialize their work, since they can focus only on one side of the problem: either RL algorithms or the development of custom environments for specific realities.

This paper puts forward a custom environment for RL agents that applies the standard defined in OpenAI Gym. To comply with it, a number of standard components must be developed [26]. The key ones are the reset function, the stepping mechanism (how the agents interact) and the reward structure.

The reset function is where the environment instances the defined JSSP, parsing its data to calculate the proper number of machines, jobs and jobs' operations that should be allocated to create a solution. The reset function is parameterizable, so that different problems can be executed. The JSSP problem instances must comply, by default, with the OR-Library format, developed by John Beasley [27]. The adoption of a standard format allows for quicker experimentation and validation of results since the optimal solutions are known. The Taillard format [28] is also supported; even if it is mostly applied on Flow-shop problems [4].

The stepping mechanism is how an agent can designate what the next action shall be. Here, the agent can select what job to plan next. There is also the possibility to work at an operational level, allocating each single operation at a time.

The reward is given to the agent as an output of the stepping mechanism, since it is the feedback from the environment concerning the latest action chosen by the agent. The reward structure that is used on the proposed implementation is in accordance with the one proposed in Figure 1.

To better explain these functions let us present an illustrated example. For that, consider that there is a problem instance, given as input to the proposed environment, that is composed by 3 jobs (T1 to T3), with 3 operations each, and 3 machines where these are executed. Instancing the problem on the environment means that the reset function would be invoked. After the problem instance is loaded the agent would be able to start making decisions; To do that, the agent would use the described stepping mechanism. If he decides that the first action shall be to allocate operation *T1.1*, the plan will change to the one represented in Figure 2.

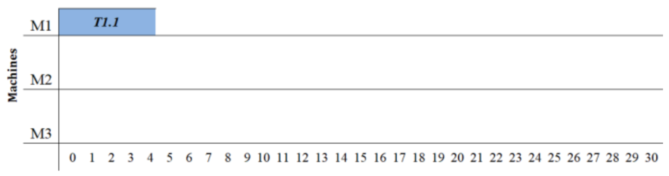

Figure 2. Plan after the action that allocates job *T1.1*

The response of the environment to the action that allocated *T1.1* would be to change to a new state (with *T1.1* on machine 1) and to provide the agent with a neutral 0.0 reward, since it was a valid allocation. Then, the agent could, for example, decide to allocate either *T2.1* or *T3.1*, as depicted in Figure 3 and Figure 4, respectively. If it tried to allocate *T1.1* again on a different machine, or T2.1 at the same time as *T1.1*, the action would be deemed as invalid, the internal state representation would stay the same and the agent would receive a -1.0 reward.

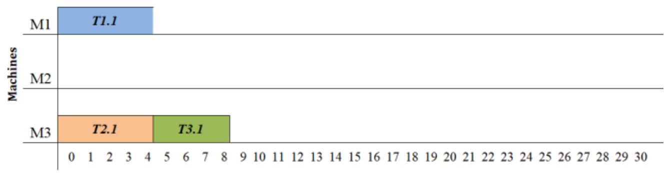

Figure 3. Plan after the second action if *T2.1* is executed first

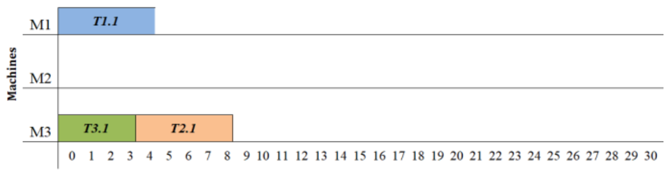

Figure 4. Plan after the second action if *T3.1* is executed first

This process of allocating an operation to a specific machine would go on until a valid plan is reached. The final plan could, perhaps, be similar to the one presented in Figure 5. When the final allocation is made the environment will compare the makespan of the plan with the optimum known value. Since this is a very simple example the reward would be of 1.0 – the distance to the optimum is lower than 250.

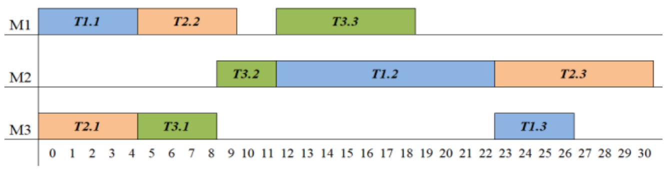

Figure 5. Plan with all jobs allocated

The rewards that are given to the agent are scaled, and fluctuate between -1 and 1. This makes it so that even if the agent can only choose neutral or negative reward actions, there should always be actions that are better than the others. When it reaches a solution, the agent would then need to be able to use long-term rewards mechanisms of modern algorithms (such as Deep Q-learning [29]) to use this information to greatly improve its performance.

Ideally, a good environment should be one where it is possible to iterate exceedingly fast and that has all the scheduling rules being employed. The rules that the proposed environment validates are related to the acceptability of the final scheduling plan. Common scheduling rules, such as the order in which the operations of the jobs are assigned and the machines where the operations are executed, are validated with each action of the agent.

Overall, the environment guarantees that the agent cannot perform invalid job allocations. This makes it easier for the training algorithms that are used to improve the performance of the agent.

The developed environment includes a set of standard agents to interact with it that follow common heuristics that planners use in real-world scenarios [30]. These are:

- After last operation – This agent will always choose as its next action the allocation of the next job immediately after the last operation on the current plan. It follows the predefined order of jobs of the JSSP (i.e. if the problem has 15 jobs, job 1 would be first and job 15 the last one allocated).

- First available slot – This agent will consider the job that needs to be planned and allocate it on the first available slot. This follows a heuristic that will lead to the reduction of idle times on the final plan [8].

- Biggest job first – This agent analyses all the jobs that must be planned and orders them considering their size, decreasingly. Allocates them on the first available slot.

- Smallest job first – Similar to the previous one, but instead orders increasingly.

- Random – this agent makes random choices on what should be planned next.

These agents do not include any type of learning or inferences from their actions, since that was not the objective; the goal of these agents is to provide a stable baseline that can be used to quickly test the environment. These tests can be made to assess the suitability of the solutions and to swiftly validate the JSSP that is used as input, i.e. if the input data that is given to the environment is not ok for some reason (e.g. jobs are not completed, missing machine information), these agents are capable of exposing it.

Nevertheless, even if the baseline agents have no learning mechanisms included, that is the main use case for this RL environment. Learning algorithms shall benefit from their existence, reducing drastically the development efforts [4].

This environment has been designed so that intelligent agents can be trained to solve JSSP using RL techniques. These agents shall be part of a complete scheduling system, composed by the artificial intelligence module, which contains the intelligent agent and the proposed environment, and by the scheduling module, which establishes all of the problem rules, supports the decoding process of academic JSSP instances and guarantees that all problem components are well-defined: machines, jobs and operations. An overall view of the proposed architecture is shown in Figure 6.

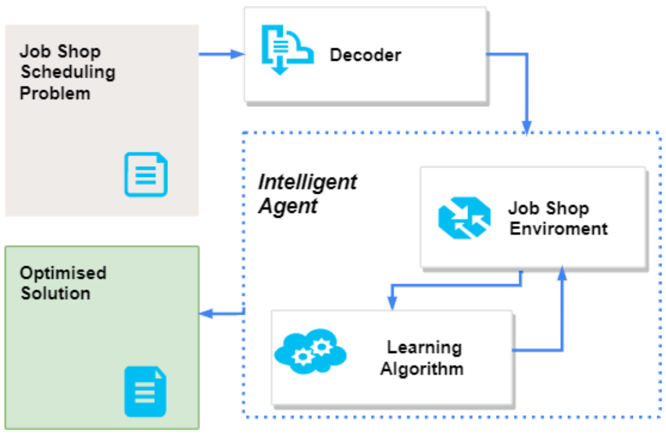

Figure 6. Diagram of the proposed architecture

V. Conclusions and Future Works

A novel RL environment that takes advantage of the latest developments of the machine learning field to solve JSSPs was presented. A JSSP has many intricacies, and small decisions can have a huge impact on the performance of a plan. The key decisions when solving a JSSP regard which jobs to allocate to which machines and in what order, considering the defined goal (e.g. comply with job deadlines) and/or performance indicators (e.g. the makespan of the proposed solution).

Current techniques that solve JSSPs are stable and well researched but are still considered to have many limitations [4]. Hence the need for machine learning techniques from one of its most promising fields; RL agents present an incredible advantage by not needing any pre-existing data, since they are capable of gaining knowledge from its own interactions with the environment (mirroring what is believed to be the human way of learning [3]).

RL solutions contain two key components: an agent that learns how to solve a problem, and an environment where the problem is modelled. This proposal details both but focuses on the environment: due to this being an original proposal and considering the complexities of the JSSP, it is essential to have a working environment develop an agent.

Every part of the proposed environment is detailed on this paper. Beginning with a formal definition of a JSSP as a RL problem (using an MDP mathematical approach) that is later used as a model for the environment, the paper also presents the technical characteristics and decisions that were made in order to achieve a quality implementation of the proposed environment.

Future work includes the research and development of agents that are capable of learning on this JSS environment. The statistical significance validation of these agents shall be done in comparison to standard benchmarks (hence the support of common formats [27], [28]). Then, the strategy is to develop a full scheduling decision support system that employs the developed agent. When the system is operational, it shall be deployed into real-world problems. At last, there shall be the disclosure, via scientific publication, of the achieved results and conclusions, including a thorough discussion of the key decisions that were taken to achieve the final system architecture.

References

[1] M. L. Pinedo, *Scheduling: Theory, algorithms, and systems*, 5th ed. Springer International Publishing, 2016.

[2] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*, Second. The MIT Press, 2018.

[3] E. A. Ludvig, M. G. Bellemare, and K. G. Pearson, “A primer on reinforcement learning in the brain: Psychological, computational, and neural perspectives,” in *Computational neuroscience for advancing artificial intelligence: Models, methods and applications*, IGI Global, 2011, pp. 111–144.

[4] B. Cunha, A. M. Madureira, B. Fonseca, and D. Coelho, “Deep Reinforcement Learning as a Job Shop Scheduling Solver: A Literature Review,” in *Hybrid Intelligent Systems*, A. M. Madureira, A. Abraham, N. Gandhi, and M. L. Varela, Eds. Cham: Springer International Publishing, 2020, pp. 350–359.

[5] K. N. McKay, F. R. Safayeni, and J. A. Buzacott, “Job-shop scheduling theory: What is relevant?,” *Interfaces (Providence).*, vol. 18, no. 4, pp. 84–90, 1988.

[6] S. A. Cook, “The complexity of theorem-proving procedures,” in *Proceedings of the third annual ACM symposium on Theory of computing*, 1971, pp. 151–158.

[7] T. Yamada, T. Yamada, and R. Nakano, “Genetic Algorithms for Job-Shop Scheduling Problems,” *Mod. Heuristi Decis. Support*, pp. 474--479, 1997.

[8] B. Cunha, A. Madureira, J. P. Pereira, and I. Pereira, “Evaluating the effectiveness of Bayesian and Neural Networks for Adaptive Schedulling Systems,” in *2016 IEEE Symposium Series on Computational Intelligence, SSCI 2016*, 2017, pp. 1–6.

[9] J. Zhang, G. Ding, Y. Zou, S. Qin, and J. Fu, “Review of job shop scheduling research and its new perspectives under Industry 4.0,” *J. Intell. Manuf.*, 2019.

[10] R. L. Graham, “Bounds on Multiprocessing Timing Anomalies,” *SIAM J. Appl. Math.*, 1969.

[11] K. Sörensen, “Metaheuristics—the metaphor exposed,” *Int. Trans. Oper. Res.*, vol. 22, no. 1, pp. 3–18, 2015.

[12] K. Sörensen, M. Sevaux, and F. Glover, “A History of Metaheuristics,” in *Handbook of Heuristics*, R. Mart í P. M. Pardalos, and M. G. C. Resende, Eds. Cham: Springer International Publishing, 2018, pp. 791–808.

[13] A. V Joshi, *Machine Learning and Artificial Intelligence*. Springer, 2020.

[14] E. L. Thorndike, “The Law of Effect,” *Am. J. Psychol.*, 1927.

[15] P. I. Pavlov, *Conditioned reflexes: an investigation of the physiological activity of the cerebral cortex*. London: Oxford University Press, 1927.

[16] J. Schulman, S. Levine, P. Abbeel, M. Jordan, and P. Moritz, “Trust region policy optimization,” in *Proceedings of the 32nd International Conference on Machine Learning (ICML-15)*, 2015, pp. 1889–1897.

[17] R. S. Sutton, “Reinforcement Learning: Past, Present and Future,” in *Simulated Evolution and Learning*, 1999, pp. 195–197.

[18] D. Silver *et al.*, “Mastering chess and shogi by self-play with a general reinforcement learning

algorithm," 2017. [Online]. Available: arXiv:abs/1712.01815.

[19] A. Nagabandi, K. Konoglie, S. Levine, and V. Kumar, "Deep Dynamics Models for Learning Dexterous Manipulation," 2019. [Online]. Available: arXiv:abs/1909.11652.

[20] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, no. 7553, p. 436, 2015.

[21] R. Kaplan, C. Sauer, and A. Sosa, "Beating Atari with Natural Language Guided Reinforcement Learning," 2017. [Online]. Available: arXiv:abs/1704.05539.

[22] T. Salimans and R. Chen, "Learning Montezuma's Revenge from a Single Demonstration," 2018. [Online]. Available: arXiv:abs/1812.03381.

[23] J. Heinrich, M. Lanctot, and D. Silver, "Fictitious self-play in extensive-form games," in *International Conference on Machine Learning*, 2015, pp. 805–813.

[24] T. Zhang, S. Xie, and O. Rose, "Real-time job shop scheduling based on simulation and Markov decision processes," in *Proceedings - Winter Simulation Conference*, 2017, pp. 3899–3907.

[25] D. Silver *et al.*, "Mastering the game of Go without human knowledge," *Nature*, vol. 550, no. 7676, pp. 354–359, Oct. 2017.

[26] G. Brockman *et al.*, "OpenAI Gym," 2016. [Online]. Available: arXiv:abs/1606.01540.

[27] J. E. Beasley, "OR-Library: distributing test problems by electronic mail," *J. Oper. Res. Soc.*, vol. 41, no. 11, pp. 1069–1072, 1990.

[28] E. Taillard, "Benchmarks for basic scheduling problems," *Eur. J. Oper. Res.*, vol. 64, no. 2, pp. 278–285, 1993.

[29] V. Mnih *et al.*, "Playing Atari with Deep Reinforcement Learning," 2013. [Online]. Available: arXiv:abs/1312.5602.

[30] A. Madureira *et al.*, "Using personas for supporting user modeling on scheduling systems," in *2014 14th International Conference on Hybrid Intelligent Systems, HIS 2014*, 2014, pp. 279–284.

Author Biographies

Bruno Cunha is an invited professor at the Institute of Engineering–Polytechnic of Porto (ISEP/IPP) and a researcher of the Interdisciplinary Studies Research Center (ISRC). He received his degree in informatics engineering in 2013 and master's degree of computer science in knowledge-based and decision support technologies in 2015 from the Institute of Engineering–Polytechnic of Porto (ISEP/IPP). He is currently pursuing his PhD studies at the University of Trás-os-Montes and Alto Douro (UTAD). His research interests involve machine learning, optimization algorithms and computational intelligence.

Ana Madureira was born in Mozambique, in 1969. She got his BSc degree in Computer Engineering in 1993 from ISEP, Master degree in Electrical and Computers Engineering–Industrial Informatics, in 1996, from FEUP, and the PhD degree in Production and Systems, in 2003, from University of Minho, Portugal. She became IEEE Senior Member in 2010. She had been Chair of IEEE Portugal Section (2015-2017), Vice-chair of IEEE Portugal Section (2011-2014) and Chair/Vice-Chair of IEEE-CIS Portuguese chapter. She was Chair of University Department of IEEE R8 Educational Activities Sub-Committee (2017-2018). She is IEEE R8 Secretary (2019-2020). She is External Member Evaluation Committee of the Agency for Assessment and Accreditation of Higher Education - A3ES for the scientific area of Informatics of Polytechnic Higher Education (since 2012). Currently she is Coordinator Professor at the Institute of Engineering–Polytechnic of Porto (ISEP/IPP) and Director of the Interdisciplinary Studies Research Center (ISRC). In the last few years, she was author of more than 100 scientific papers in scientific conference proceedings, journals and books.

Benjamim Fonseca is an Assistant Professor with Habilitation at the University of Trás-os-Montes and Alto Douro (UTAD) and researcher at INESC TEC, in Portugal. His main research and development interests are collaborative systems, mobile accessibility and immersive systems. He authored or co-authored over a hundred publications in these fields, in several international journals, books and conference proceedings, and participates in the review and organization of various scientific publications and events. He is also co-founder and CEO of 4ALL Software, UTAD's spin-off that works in the development of innovative software, implementing complex digital platforms and interactive solutions.