Reinforcement Learning for Job Shop Scheduling Problems

Ricardo Miguel Alves Magalhães
ricardo.m.a.magalhaes@tecnico.ulisboa.pt

Instituto Superior Técnico, Universidade de Lisboa, Lisboa, Portugal

January 2021

Abstract

There is a revolution going on in the way industries work. The large abundance of data and the exponential growth of computational power have unchained Artificial Intelligence's potential. This new powerful tool gained popularity amidst the advent of Industry 4.0 and comes with the promise of making factories smart, more productive and efficient. The method proposed in this thesis applies Deep Q-Learning to create an intelligent system capable of solving the Dual Resource Constrained Flexible Job Shop Scheduling Problem. A novel Encoder-Decoder neural network architecture with Attention Mechanism is presented. This architecture allows the network to be fed more thorough and meaningful problem information, as well as having more precise and powerful output commands. In order to diminish the computational cost of this technique, the output commands are selected as in Pointer Networks architectures. The agent developed achieved promising results, even having just been scarcely trained. These results show that, with an optimized implementation of the method and a more thorough training procedure, this technique holds the potential of challenging current state-of-the-art meta-heuristic algorithms in solving this kind of problems. Finally, future work might consist in adding additional constraints to the problem for application in the Industry. The possibility of having different processing times for the same operation, depending on the allocated resource pair, is discussed. Also, a redesigned concept of the proposed method is presented.

Keywords: Job-Shop Scheduling, Reinforcement Learning, Deep Q-Learning, Encoder-Decoder, Attention Mechanism, Pointer Networks

1. Introduction

We are living amidst a revolution in the way industries work and companies do business, as Industry 4.0 is on the rise and taking its first steps. Even though some might still dismiss it as a fancy marketing buzzword, there are huge transformations happening right now in the manufacturing processes of some of the world leading companies, that definitely make these new systems worthy of our attention and consideration [10].

The large abundance of data and the exponential growth of computing power have unchained AI's potential and made its costs significantly decrease. The evidence seems to point out that we are on the verge of an immense upshift of the field's development and investment, and so a vast progress and fierce solutions are expected in the next few decades [12]. Areas where algorithmic problem-solving used to be found as extremely complex are now, with the introduction of AI, getting to consider these methods as feasible and economically viable [14]. The opportunities are immense and difficult to thoroughly grasp and so are the uncertainties about the changes it will create to our future. One thing is for sure: AI stands out as one of the most promising tools for companies to leverage their future. And perhaps to lose the trail of this transformation might cause irreversible damage to a company competitiveness in the market.

1.1. Dual Resource Constrained Flexible Job Shop Scheduling

Scheduling is the process of allocating scarce resources to perform a number of competing tasks over time [6]. It is a decision making process of great importance in the fields of manufacturing and production [13]. This important role derives from the need for survival that companies find in an extremely competitive economic scenario, where profit margins are getting smaller and smaller [6]. Optimized production schedules can give an enterprise a more efficient utilization of resources, which has a considerable impact on their production's capacity of meeting deadlines while maximizing profits. A study as shown that the lack of quality in planning and scheduling can reduce productivity by 5 percent [14]. Quite often, inventories are increased to improve supply reliability and uphold stable production, which represents an increase in production

costs. Excellent planning and scheduling is thus a significant competitive advantage.

As the manufacturing industry reveals an increasing reliance on agility and flexibility, with the expansion of product customization, which lead to smaller lot sizes and shorter cycle times, the different kinds of machine and human resources must be considered and effectively managed to assure quick responses to the market demands. The Dual Resource Constrained Flexible Job Shop Scheduling Problem (DRC-FJSSP) effectively models these dynamics. It consists of 2 routing optimization subproblems: job scheduling and resource dispatch [9]. The route that each job has to follow may not be predetermined. For example, there may be several machines capable of executing the exact same operation and several workers capable of operating the same machine. The goal when solving this problem is to optimize a certain performance criteria, like minimizing the production cost, the makespan or the weighted tardiness, while respecting the problem constraints. A numerical model of the DRC-FJSSP is presented in section 3.1.

The DRC-FJSSP has been traditionally solved using analysis and simulation approaches. However, some details of the DRC-FJSSP are difficult to model for analysis and, being an NP-hard problem [5], simulations have an unbearable computational cost. Therefore, there has been a trend towards the use of meta-heuristic methods, since they have proved to be capable of quickly find near optimal solutions [9].

The idea that led to this thesis was to use Deep Q-Learning to create an intelligent system capable of generating quality solutions to the DRC-FJSSP. The developed agent was designed to be flexible, so that it can solve a wide range of problems and be suitable for applicability to the Industry. That implies that the generated solutions have to be reached in a reasonable amount of computational time. Finally, this work aimed to extend the limits of the application of Deep Learning to the DRC-FJSSP and pave the way for future research.

1.2. State-of-the-Art Metaheuristics

In the research conducted for this thesis a set of meta-heuristics were found to constitute the backbone of state-of-the-art techniques to solve the DRC-FJSSP. A Branch Population Genetic Algorithm was used to try to minimize the makespan and the cost of the DRC-FJSSP [9]. The novelty here is in introducing a branch population to accumulate and transfer evolutionary experience, strengthening the population diversity and accelerating convergence. A permutation-based search operator is used to improve the thoroughness of global exploration and a greedy selection of the best solutions is used for local exploitation.

A Shuffled Multi-Swarm Micro-Migrating Birds Optimizer tries to minimize the makespan of a Multi-Resource Constrained Flexible Job Shop Scheduling Problem [4], similar to the DRC-FJSSP but with more than two resource constraints. The algorithm forms a number of micro-swarms, each of which performs its own Migrating Birds Optimizer independently. It has been found that a genetic algorithm with a small population of three individuals is sufficient to converge, irrespective of having different lengths of chromosomes involved [7]. A random shuffle process is periodically applied to propagate the knowledge acquired by the micro-swarms. Also, there is a renewing process of the population, based on the aging phenomenon of life, that promotes the diversity of the population. Two different types of permutation-based search operators are used that try to efficiently balance global exploration and local exploitation.

A Knowledge Guided Fruit Fly Optimization Algorithm (KGFOA) proved to be effective in solving the DRC-FJSSP [17]. Two types of permutation-based search operators were used for exploration and optimized by being combined with a knowledge-guided search stage. In summary, this knowledge-guided search utilizes the experience provided by the best solutions found so far to enhance the probabilities of choosing of operations and resource assignments that have proved to work in the past. Finally, a greedy selection of the best solutions takes place. This whole procedure is applied with the goal of minimizing the solution makespan. Since this is the method with the most recognition from this set, it was the algorithm chosen for comparison with the solution proposed in this thesis.

The remaining of this paper is organized as follows. Section 2 there is an introduction to the Encoder-Decoder architecture with Attention Mechanism and Pointer Networks. Section 3 details a numerical implementation of a DRC-FJSSP, as well as the steps that led to the development of the proposed solution, regarding the usage of Deep Q-Learning and the implemented RL algorithm. Results are presented and discussed in section 4. Section 5 concludes the paper and presents future research steps.

2. Background

In Deep Q-Learning a deep neural network is used to approximate the Q-value function. The Encoder-Decoder with attention mechanism was chosen as the NN architecture. This choice will allow the agent to receive inputs with variable sizes, a required feature for it to be able to solve problems with any number of operations to schedule, with-

out the need to generate a size-fixed simplified representation of the input information. In figure 2, a first layout of the chosen architecture is given. A step by step presentation of it will be given in the following sections.

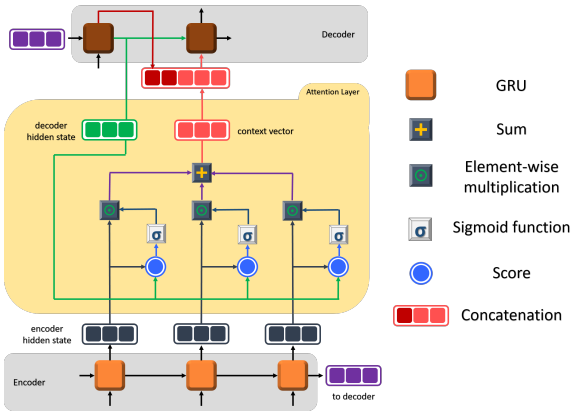

Figure 1: An overview of the Encoder-Decoder with Attention mechanism architecture.

2.1. Encoder-Decoder

The fundamental idea behind an Encoder-Decoder architecture is to have two different RNN's: one which sequentially receives the input information and summarizes it in a vector representation of fixed dimensionality (encoder) and another which receives this vector and sequentially generates the output (decoder). This means that the only information the decoder receives is the last encoder hidden state. A representation of this architecture is given in figure 2.1.

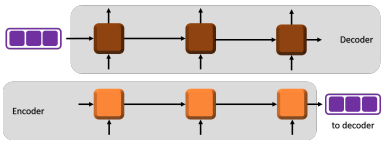

Figure 2: A basic Encoder-Decoder architecture.

The problem with it is that, when the input length gets really large, it is extremely difficult to sufficiently summarize all the input data in a single fixed-sized vector. In other words, this might lead the agent to forget input information. That is why an attention layer is introduced.

2.2. Attention Mechanism

The attention layer is an interface that allows the Decoder to access information about all the Encoder hidden states at all times. Moreover, it highlights useful parts of that input data in order to let the model focus on those key areas and learn the connections between them. This way, the model can effectively remember and process long input sequences [8].

There are two types of attention mechanisms: one that uses all encoder hidden states (global attention) and another that uses only a subset of them (local attention). All throughout this thesis, global attention was the type of mechanism used and referred to as "attention".

During the steps on the attention layer, a set of attention weights is calculated, one weight for each encoder hidden state. These attention weights are the ones who define how relevant each input is for the output of the current time step and so are used to derive the context vector fed as input to the decoder.

One of the fields where the usage of these Encoder-Decoder with attention mechanism architectures are popular is Neural Language Processing, or more specifically the translation task. A representation of the attention weights is given in figure 2.2, where the darker lines represent higher weight values.

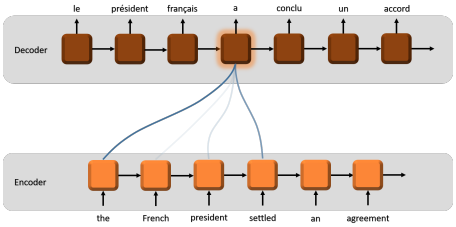

Figure 3: An intuitive example about the influence of the attention weights.

The first step which takes place in the attention layer is the calculation of a score between the current decoder hidden state and all the encoder ones. Different operations between the vectors can be applied to calculate this score. One option is to calculate the dot product between the two vectors. In this thesis, each of the weights u_v^d, the attention weight in the decoder step d relative to the v-th encoder hidden state, was calculated according to equation 1.

$$u_v^d = w^T tanh(W_1 e_v + W_2 d_d),\ v \in (1, ..., n) \quad (1)$$

where w, W_1 and W_2 are learnable weights, e_v is the v-th encoder hidden state and d_d is the d-th decoder hidden state.

Then all the scores go through a softmax function, which is a generalization of the logistic function for multiple dimensions. After applying softmax, each score will be in the interval $(0, 1)$ and the components will add up to 1. After that, each encoder hidden state is multiplied by its own softmaxed score. And finally, all the resulting vectors are summed up, generating the context vector. This set of steps ends up being a linear combination the

encoder hidden states, where the constants that are multiplied to the each hidden vector is its own score. The calculations made during these steps are presented in equation 2.

$$a_v^d = softmax(u_v^d),\ v \in (1, ..., n) \tag{2a}$$

$$d_d^{'} = \sum_{v=1}^{n} a_v^d e_v \tag{2b}$$

A final step is left, during which the context vector is fed into the decoder. Although there are different common ways of doing this, in this thesis the Bahdanau's method was chosen [1]. According to this method, the input to the next decoder step is the concatenation between the output from the previous decoder time step and context vector from the current time step.

Despite its complexity, this network architecture can still be trained using Backpropagation. The algorithm will change the weights in the RNNs and in the score function in order to ensure that the outputs will be close to the ground truth. These weights will affect the encoder hidden states and decoder hidden states, which in turn affect the attention scores.

At last, there is one small detail still worth highlighting. It is possible to run as many steps of the decoder as needes. In other words, for the translation task, the agent can output as many words (one by one) as needed. However, the output of a GRU is constant in size. This means, that size of the dictionary of words that can be used is fixed. It cannot be adapted from problem to problem. This constitutes a limitation that Pointer Networks architectures aim to solve.

2.3. Pointer Networks

Pointer Networks introduce a very simple modification to the attention model that make it possible to solve combinatorial optimization problems where the output dictionary size depends on the number of elements in the input sequence [15].

It does that by using the attention weights as pointers to the input elements and taking them as the output of each decoder step. The softmax function generates these weights such that they will add up to 1, which means that they can be interpreted as probabilities, just like in equation 3.

In other words, it uses the attention weights as pointers to the input elements.

$$u_v^d = w^T tanh(W_1 e_v + W_2 d_d),\ v \in (1, ..., n) \tag{3a}$$

$$P(C_d | C_1, ..., C_{d-1}, \mathcal{P}) = softmax(u^d) \tag{3b}$$

Here $\mathcal{P} = \{P_1, ..., P_n\}$ is the input sequence and $C^{\mathcal{P}} = \{C_1, ..., C_{z(\mathcal{P})}\}$ is the output sequence.

So C_d is the output of the d-th decoder step. It is a vector with the attention weights of that step (one weight for each encoder hidden state).

3. Implementation

3.1. Numerical Model

In a DRC-FJSSP there is a set of n jobs $J = \{J_1, J_2, ..., J_n\}$ to be processed at a set of m machines $M = \{M_1, M_2, ..., M_m\}$ operated by a set of w workers $K = \{K_1, K_2, ..., K_k\}$. Each job has a predefined sequence of n_i operations $O = \{O_{i,1}, O_{i,2}, ..., O_{i,n_i}\}$. Let p_{ij} be the processing time of the operation $O_{i,j}$. Operation $O_{i,j}$ can only be processed at one machine out of a set of $M_{i,j}$ eligible machines. Each machine can process only one operation at a time and there is no preemption, which means that since an operation has started it cannot be stopped until it is finished. Each worker K_k can only operate a subset of M, for which K_k is an eligible worker. Thus, there is an eligibility matrix E_{mk}, where the element (m,k) is a 1, if K_k is an eligible worker for M_m, or, otherwise, it is a 0. All jobs, machines and workers are available at time 0. The goal is to minimise the maximum completion time, the makespan C_{max} by assigning a compatible machine and an eligible worker to each operation as well as arranging the processing order of operations on each machine.

Let st_{ij} be the starting time of $O_{i,j}$, and rt_{mk} be the ready time of machine M_m operated by worker K_k, and N be a large enough number. Mathematically, the DRC-FJSSP with makespan minimisation can be formulated as follows:

$$Min\ C_{max} \tag{4}$$

Subject to:

$$st_{i(j+1)} \geq st_{ij} + \sum_m \sum_k p_{ij} x_{ijmk},\ J_i \in J,$$
$$j = 1, 2, ..., n_i - 1, M_m \in M_{i,j},\ E_{mk} > 0 \tag{5}$$

$$st_{i^{'}j^{'}} + (1 - \zeta_{ijm-i^{'}j^{'}m})N \geq s_{ij} + \sum_k p_{ij} x_{ijmk},$$
$$J_i \in J,\ j = 1, 2, ..., n_i,\ M_m \in M_{i,j},\ E_{mk} > 0 \tag{6}$$

$$rt_{m^{'}k} + (1 - \xi_{mk-m^{'}k})N \geq rt_{mk},$$
$$K_k \in K,\ E_{mk},\ E_{m^{'}k} > 0 \tag{7}$$

$$rt_{mk} + (1 - x_{ijmk})N \leq st_{ij},\ J_i \in J,$$
$$j = 1, 2, ..., n_i,\ M_m \in M_{i,j},\ E_{mk} > 0 \tag{8}$$

$$\sum_m \sum_k x_{ijmk} = 1,\ J_i \in J,$$
$$j = 1, 2, ..., n_i,\ M_m \in M_{i,j},\ E_{mk} > 0 \tag{9}$$

$$x_{ijmk} = \begin{cases} 1, & \text{if } O_{i,j} \text{ is processed on } M_m \\ & \text{operated by } K_k \\ 0, & \text{otherwise} \end{cases} \quad (10)$$

$$\zeta_{ijm-i'j'm} = \begin{cases} 1, & \text{if } O_{i,j} \text{ is processed before} \\ & O_{i'j'} \text{ on } M_m \\ 0, & \text{otherwise} \end{cases} \quad (11)$$

$$\xi_{mk-m'k} = \begin{cases} 1, & \text{if } M_m \text{ is operated before} \\ & M_{m'} \text{ by } K_k \\ 0, & \text{otherwise} \end{cases} \quad (12)$$

where equation 5 ensures that the precedence constraints are not violated; equation 6 guarantees that a machine can process only one operation at a time; equation 7 ensures that a worker can operate one machine at a time; equation 8 ensures that an operation cannot start unless the assigned resources are ready; equation 9 guarantees that each operation is assigned to only one compatible machine operated by one eligible worker.

3.2. Proposed Solution

A great inspiration for this thesis was retrieved from the work in [16] and in [11], regarding the use of RL to solve the DRC-FJSSP and the RL algorithm, respectively. In the following subsections, both these aspects are discussed as the proposed solution is presented.

3.2.1 Solve the DRC-FJSSP using RL

In [16], a novel approach, using Deep Q-Learning, was proposed for solving the DRC-FJSSP. In their work, they initialized a schedule according to a set of rules and ran a step by step optimization process. At each step, the agent would receive a representation of the current schedule, the state, as input and output a choice between 2 possible actions: Move and Reassign Pool. Each of them was an heuristic that the agent could choose to apply in order to update the state. Soon, they realized that a full schedule representation, a vector with all the problem variables, could not be used as a state, since its size would vary with the number of jobs and resources of a scheduling problem and the number of inputs of a NN has to be fixed. Therefore, a 30 feature state representation was used. This arbitrarily chosen set of features was supposed to summarize all the state information.

In the early research of this thesis, this was identified as a plausible source of the limited optimization results achieved, since the agent's intelligent intuition had limited power and information to act upon. Allowing the intelligent agent to analyse, move and assign individual operations would give it total control and thus unchain its true potential. In order to do that, one would need to able to feed the NN a variable number of input features and let it choose between a variable number of possible actions. For example, consider a simple problem with n jobs. As an input, the agent needs to receive n input vectors, each one of them with information regarding each individual job. Whereas, as an output, it needs to be able to choose to move one job, out of the whole set, into a new position in the schedule. Thus, it would need to output a set of n values, being each of those values the score of choosing to move each job. Also, unlike in [16], the initial schedule is randomly generated and only feasible states are allowed throughout the optimization process. This means that the initial state is feasible and every move made by the agent needs to keep it like that, to be allowed.

In order to do this, the encoder-decoder with attention mechanism was used. It is a RNN architecture, typically used for Natural Language Processing, that is able to receive inputs with variable sizes, deliver outputs with variable sizes and have a high quality memory, even for long input sequences. These features make this architecture potentially powerful for the DRC-FJSSP. The overview of this proposed solution is given in figure 3.2.1.

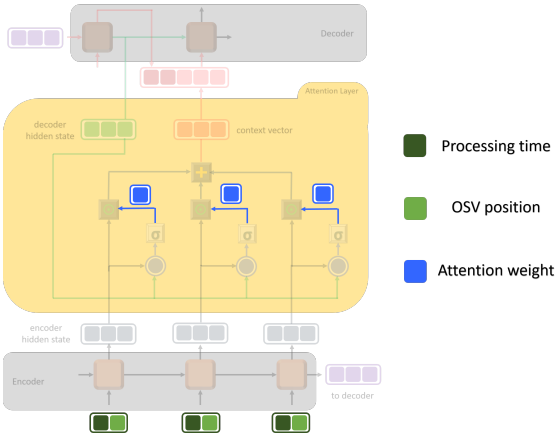

Figure 4: Overview of the proposed RNN architecture, with highlight to its inputs and outputs.

This architecture is quite complex on its own, so there was an attempt to simplify the inputs to the maximum. Therefore, there will be a number of encoder steps equal to the number of operations to schedule. At the v-th encoder step, information about the v-th operation is submitted to the agent. That information is composed by a 2 component vector. The first component represents the normalized processing time of that operation. Whereas,

Table 1: Example of OSV, WAV and MAV schedule encoding vectors.

OSV	4	1	7	9	5	10	6	2	8	3
MAV	1	3	1	2	2	3	1	1	3	3
WAV	2	1	2	2	2	1	1	2	1	1

the second component represents the normalized position of that operation in the OSV.

The Operation Sequence Vector (OSV) is one of three different vectors responsible for encoding the schedule information. The other vectors are the Machine Allocation Vector (MAV) and the Worker Allocation Vector (WAV). This encoding scheme is similar to the ones used in many state-of-the-art methods and it aims to simplify schedule representation and manipulation.

All of these vectors have a number of components equal to the number of operations to schedule. The OSV gives us the order by which operations are put in the schedule, which means that the v-th component of the vector is the i-th operation to be scheduled. The v-th component of the MAV and the WAV tell us what is the machine and worker allocation for the operation in the v-th component of the OSV. An example of this encoding scheme is presented in table 1.

During the decoding process, operations are put in the schedule one at a time, following the order specified in the OSV. They are put at the minimum feasible time, which will logically depend on the availability of the allocated worker and machine. For example, according to table 1, the first component in the OSV is operation number 4. So, this would be the first operation to be put in the schedule. It would be allocated to machine number 1, operated by worker number 2. Next, the second component of the OSV, operation number 1, would be put in the schedule, allocated to machine number 3, operated by worker number 1. And so on, until operations are put in the schedule.

Finally, as can be seen in figure 3.2.1, the decoder outputs will not be considered as the agent outputs. In fact, the attention weights will be considered as the output of the network, as in a Pointer Network. This is done for 2 main reasons: computational power and intuition.

First of all, computational power. The number of outputs our agent needs to deliver is equal to the squared value of the number of operations to schedule. It has n operations to choose from and n positions in the OSV to move them to, since the agent is allowed to "move" an operation to the same position it is in. So, all in all, it has n^2 possible moves to choose from. Since this number varies with the problem dimension, the agent would need to output the score of each of this moves one by one in the decoder, which means it would need n^2 decoder steps. At each decoder step, there are n attention weights calculated, one for each encoder hidden state. This means that, if they are used as the network's outputs, there is only the need to run n decoder steps to find the n^2 score values. This means that decoder's GRU is ran a much smaller number of times, especially when the problem size increases a lot. Therefore, a lot of computational effort is saved.

Secondly, intuition. It was shown that the attention weights are score values that basically evaluate the importance of each input the encoder received. In a way, they be can pictured as the "amount of time" the agent would "look" at each of these inputs before deciding what would be the best output to deliver. So the word intuition is used in the sense that it is reasonable to assume that if the agent is concerned a lot ("looks a lot of time") with an operation, then that operation is probably the most critical one to improve the current solution quality. So, the operation with the highest attention weight is probably the most suitable choice to be moved. So, there are n decoder steps, each one with n attention weights. First, the highest attention weight of them all is located. If it is the v-th attention weight of the d-th decoder step, then the agent chooses to move the v-th operation to the d-th position in the OSV. Of course, the justification behind this reasoning is not rigorous, but it is an assumption that can save a lot of computational effort and, so, it is definitely worth testing and evaluating the results.

The simplified inputs given to the agent do not contain any information regarding the workers and machines available for allocation. That is why the agent only manipulates the OSV. The allocation of machines and workers will be made at the schedule decoding phase, according to the Earliest Feasible Time heuristic. This means that each operation will be allocated to the worker-machine pair that can process it the earliest. If there are more than a single pair available at the same earliest time, then the first of them is chosen. Worker-machine pairs (M_m, K_k) are listed according to equation 13. So, the first would be the pair with the smallest index. This was made in order to ease programming implementation.

$$Pair\ index = (m-1)n_{workers} + k \qquad (13)$$

3.2.2 RL algorithm

In [11], a deep Q-network was presented. Using end-to-end reinforcement learning, the agent was able to achieve a level comparable to that of a professional human games tester across a set of 49 Atari games, using the same algorithm, network archi-

tecture and hyperparameters. Motivated by those impressive results, the same reinforcement learning algorithm was followed in this thesis, like described in algorithm 1.

Algorithm 1: Deep Q-Learning with experience replay

```
Initialize replay memory D to capacity DN ;
Initialize action-value function Q with
  random weights w ;
Initialize target action-value Q̂ with weights
  w⁻ = w ;
for episode = 1, ..., M do
    Initialize sequence s_1 = x_1 and
      preprocessed sequence φ_1 = φ(s_1) ;
    for t = 1, ..., T do
        With probability ε select a random
          action act_t ;
        otherwise select
          act_t = argmax_act Q(φ(s_t), act; w) ;
        Execute action a_t in simulation and
          observe reward r_t and state x_{t+1} ;
        Set s_{t+1} = s_t, act_t, x_{t+1} and
          preprocess φ_{t+1} = φ(s_{t+1}) ;
        Store transition (φ_t, act_t, r_t, φ_{t+1}) in
          D ;
        Sample random minibatch of
          transitions (φ_v, act_v, R_v, φ_{v+1}) from
          D ;
        if episode terminates at step v+1
          then
            targ_v = R_v ;
        else
            targ_v =
              R_v + γ max_act' Q̂(φ_{v+1}, a'; w⁻) ;
        end
        Perform a gradient descent step on
          (targ_v − Q(φ_v, act_v; w))² with
          respect to the network parameters w
          ;
        Every C steps reset Q̂ = Q ;
    end
end
```

All in all, during training the agent will try different moves and save that information (initial state, move and final state) in a memory, together with the respective reward. Exploration and exploitation are balanced according to the ϵ-greedy algorithm. This means that the agent will take a random action with probability ϵ, while the rest of the times it will choose the action elected by the output of the RNN. This ϵ value was changed linearly between 100%, in the beginning, and 10%, after 80% of the epochs. During the rest of the epochs, the ϵ is of 10%. This is a typical range of values when applying ϵ-greedy.

Each epoch is composed by a number of episodes. Every episode, the agent receives a new problem and does $3 * n_{operations}$ moves, where $n_{operations}$ is the number of operations to schedule for that problem. This number of moves is derived in appendix A of the thesis, while the number of episodes is chosen empirically.

Then, a mini-batch is generated by randomly select 32 instances from the memory. Different mini-batch sizes were tested, but the best results were found for that value. After that, a target value $targ_v$ is calculated for each of those instances according to the if statement in 1. The γ value is a parameter that is common practice to establish as 0.99, even though it might be optimized to the problem at hand. Finally, a gradient descent is performed trying to minimize the squared error between the target values and the predictions (output moves) dictated by the RNN.

The rewards play a huge part in the success of a RL method and they usually depend and need to be adapted to the problem at hand. In this thesis implementation, some different reward combinations were tried, until a combination was found that delivered good results. In the end, it was found that to deliver strong rewards when good moves were made was better for the algorithm to learn then to deliver penalties for bad movements. So a reward of 0.05 was given for every unit of time a move could improve the solution makespan relative to the best makespan found so far. Otherwise, if a new best solution was not found, a small penalty of 0.01 was given. Furthermore, a growing penalty of 0.001 times the number of moves so far was given in order to incentive the agent to find good solutions fast. At last, the agent was given the possibility to suggest moves that do not change the schedule. It happens when the agent wants to move one operation to a position in the OSV it is already in. In that case, it was considered that the agent is stating that the schedule is already optimized. No penalties are given if that is the chosen move.

After the training process, hopefully the Agent will be ready to autonomously generate quality solutions for the DRC-FJSSP. Those solutions are generated through a step by step optimization process. At each step the agent receives as an input information about the current solution state, just as it did during training, and outputs the selection of a move operation. A rule was established to put an end to the agent's optimization process. A schedule is considered to be optimized when the agent chooses a move that does not change the OSV. When that happens, the best schedule found so far, the one with the lowest makespan, is considered to be the optimized solution. Additionally, the agent

only has a limited number of moves to optimize the solution, equal to 50% of the number of operations to schedule for the problem at hand. After those moves, the best schedule found so far, the one with the lowest makespan, is considered to be the optimized solution. The 50% threshold was defined empirically as a trade-off between the minimization of computational time and the maximization optimization performance.

4. Results

An agent was trained for application in a widely used benchmark dataset, the MK1-10 [2]. This benchmark has 2 big constraints that are common in a DRC-FJSSP. The first one of them is that jobs can have multiples operations, that need to be executed in a precise sequence. The second one is that only a subset of the machines is eligible to execute each operation and only a subset of the workers is able to operate each machine. The training set was composed of 10 different problems, each with 10 operations to schedule, with processing times between 2 and 18 units of time, randomly generated. Unlike in the test dataset, each machine was eligible for any operation and each worker could operate any machine. Finally, jobs were composed by a single operation. For the hyperparameters, a combination that achieved faster but steady convergence during training was used. For the same reason, a learning rate of 0.01 was chosen, together with no target network $\hat{Q}$ weights update, a batch size of 32 and the reward system described earlier. The KGFOA was chosen as a state-of-the-art meta-heuristic for comparison and implemented as described in [17] for 1000 generations. Both models were implemented in Python and ran in a Google Colab environment, with GPU enabled. The results of the simulation are presented in table 5.

For half of the test instances (MK's 4, 5, 6, 8 and 10) the KGFOA performed considerably better than the Agent. However, for the rest of them, the results from both methods were quite competitive. The Agent was even able to surpass the KGFOA's performance for MK2. It is true that on average the KGFOA had a superior performance, but these are still impressive results given that the Agent had not been exposed to the additional constraints of the MK1-10 dataset during training. In other words, the Agent had never had to solve a problem before where jobs had multiple operations, with a predefined sequence of execution, or where there were worker-machine pairs that were not eligible for allocation. This means that a scarcely trained agent can still occasionally compete with a state-of-the-art technique when solving some problems way more complex than the ones it was trained with.

Additionally, the execution times of the Agent were more than 60 times lower than the KGFOA. One may point out that for a fair analysis, the Agent and the KGFOA results should be compared for a similar execution time. However, for the MK1-10 instances, the KGFOA execution times, even tough much higher than the Agent's, were still within a reasonable range. However, for larger problems that probably would not be the case anymore and so the Agent's celerity could come as a decisive advantage.

There is another feature of the Agent worth highlighting, relative to the execution time of the optimization process. As mentioned in section 3.2.2, the agent can put an end to the optimization process if it identifies the current schedule as being fully optimized. This skill is supposed to be acutely develop at least for a fully trained agent. However, for the scarcely trained agent being tested, it shall not come as surprise that most of the times it fails to recognize that a schedule is fully optimized. It would already be remarkable if it could even achieve such a schedule. More often than not, the agent will finish the optimization process when it reaches the maximum number of moves stop criterion. Therefore, it is expected that the execution times sharply decrease for a thoroughly trained agent.

5. Conclusions

The research conducted on this master thesis allowed the development of a novel Deep Q-Learning method for solving the DRC-FJSSP. The idea of applying an Encoder-Decoder NN architecture with an Attention Mechanism, typically used when solving Neural Language Processing problems, really proved to work. Even with just a scarce number of training cycles across a very basic training dataset, the agent designed was able to achieve some competitive results comparatively with a state-of-the-art meta-heuristic when applied to a benchmark dataset. The agent proved to be flexible in dealing with some real-world Industry constraints and was able to generate the solutions in a reasonable amount of time, dozens of times faster than the KGFOA.

The most significant drawback of this implementation has to be the computational effort required to train the NN. The major bottleneck of the work developed in this thesis was indeed the lack of adequate computational resources. This heavy computational requirements are typical in the development of Deep Learning systems. However, as pointed out in the introduction of this thesis, the computational power available in the market has been growing exponentially. If one or two decades ago the amount of computational power required to train a Deep NN would only be available to the

Table 2: Simulation results of the Agent and the KGFOA for the MK1-10.

	MK1		MK2	
	Agent	KGFOA	Agent	KGFOA
Job Dimension (%)	319.07	**162.79**	**109.88**	231.98
Job Difference (‰)	1577.47	**804.83**	**666.10**	1406.21
Min Finish time (units)	53	55	66	71
Max Finish time (units)	66	**60**	**69**	74
Lower Bound (units)	54	54	65	65
Distance to Bound (%)	22.22	**11.11**	**6.15**	13.85
Execution Time (s)	**1.26**	65.46	**1.34**	75.27
	MK3		MK4	
	Agent	KGFOA	Agent	KGFOA
Job Dimension (%)	1523.67	**734.71**	640.80	**537.36**
Job Difference (‰)	2537.50	**1223.58**	2463.10	**2065.47**
Min Finish time (units)	285	311	107	114
Max Finish time (units)	408	**328**	129	**123**
Lower Bound (units)	254	254	92	92
Distance to Bound (%)	60.63	**29.13**	40.22	**33.70**
Execution Time (s)	**3.02**	262.98	**1.85**	131.87
	MK5		MK6	
	Agent	KGFOA	Agent	KGFOA
Job Dimension (%)	683.17	**328.45**	1637.44	**992.65**
Job Difference (‰)	9156.44	**4402.13**	5347.98	**3242.04**
Min Finish time (units)	341	324	121	155
Max Finish time (units)	359	**332**	207	**169**
Lower Bound (units)	307	307	111	111
Distance to Bound (%)	16.94	**8.14**	86.49	**52.25**
Execution Time (s)	**2.23**	202.06	**2.93**	282.48
	MK7		MK8	
	Agent	KGFOA	Agent	KGFOA
Job Dimension (%)	405.03	**339.85**	1352.33	**949.40**
Job Difference (‰)	616.76	**517.51**	3612.11	**2535.86**
Min Finish time (units)	281	290	646	586
Max Finish time (units)	312	**305**	688	**637**
Lower Bound (units)	269	269	517	517
Distance to Bound (%)	15.99	**13.38**	33.08	**23.21**
Execution Time (s)	**2.75**	142.46	**4.61**	534.35
	MK9		MK10	
	Agent	KGFOA	Agent	KGFOA
Job Dimension (%)	1026.24	**933.42**	1304.43	**967.27**
Job Difference (‰)	3179.76	**2892.14**	3782.19	**2804.60**
Min Finish time (units)	627	636	449	465
Max Finish time (units)	667	**655**	528	**487**
Lower Bound (units)	535	535	370	370
Distance to Bound (%)	24.67	**22.43**	42.70	**31.62**
Execution Time (s)	**4.84**	549.72	**4.74**	486.88

market's technological giants, nowadays an average startup budget would probably be enough to invest in those resources. Moreover, the advent of cloud computing diminished even more the price of access to those resources. If that would not be enough, just recently Chinese scientists announced a technological breakthrough, claiming that they had developed a quantum computer capable of performing certain computations nearly 100 trillion times faster than the world's most advanced supercomputer [3]. All in all, the rate at which the computational power available for everyone is growing does not seem to decay, so it should not be long until it becomes easy for everyone to train a Deep NN in their own computer. But even now the required computational power should be available at a reasonable budget.

It was impossible to evaluate the full capabilities of the proposed solution due to the lack of computational resources. The complex neural network architecture developed would need to be exposed to at least hundreds of different problems and hundreds of thousands of training cycles, before one could consider it properly trained. Therefore, as future work, before trying different approaches it would be interesting to evaluate how well can this model operate after a thorough training procedure and compare the results obtained with the state-of-the-art techniques.

References

[1] D. Bahdanau, K. Cho, and Y. Bengio. Neural machine translation by jointly learning to align and translate, 2014.

[2] P. Brandimarte. Routing and scheduling in a flexible job shop by tabu search. *Annals of Operations Research*, 41:157–183, 1993.

[3] S. Chen. Chinese scientists claim breakthrough in quantum computing race. Accessed at 31/12/2020.

[4] L. Gao and Q.-K. Pan. A shuffled multi-swarm micro-migrating birds optimizer for a multi-resource-constrained flexible job shop scheduling problem. *Information Sciences*, 372, 08 2016.

[5] M. R. Garey, D. S. Johnson, and R. Sethi. The complexity of flowshop and jobshop scheduling. 1(2):117–129, May 1976.

[6] G. Georgiadis, A. Elekidis, and M. Georgiadis. Optimization-based scheduling for the process industries: From theory to real-life industrial applications. *Processes*, 7:438, 07 2019.

[7] D. Goldberg. *Genetic Algorithms in Search Optimization and Machine Learning*. Addison-Wesley, Menlo Park, CA, 1988.

[8] R. Karim. Attn: Illustrated attention. Accessed at 07/11/2020.

[9] J. Li, Y. Huang, and X. Niu. A branch population genetic algorithm for dual-resource constrained job shop scheduling problem. *Computers Industrial Engineering*, 102, 10 2016.

[10] B. Marr. What is industry 4.0? here's a super easy explanation for anyone, 09 2018. Accessed at 28/10/2020.

[11] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2015.

[12] M. Morales. *Grokking Deep Reinforcement Learning*. Manning Publications, 2020.

[13] M. L. Pinedo. *Scheduling: Theory, Algorithms, and Systems*. Springer International Publishing, 5th edition, 2016.

[14] T. G. W. C. S. F. Roman Hipp, Christian Fiebig. Automation of production planning enhanced by ai, 2019.

[15] O. Vinyals, M. Fortunato, and N. Jaitly. Pointer networks. *Proceedings of the 28th International Conference on Neural Information Processing Systems - Volume 2*, page 2692–2700, 2015.

[16] W. Zhang and T. Dietterich. Solving combinatorial optimization tasks by reinforcement learning: A general methodology applied to resource-constrained scheduling. 06 2001.

[17] X.-l. Zheng and L. Wang. A knowledge-guided fruit fly optimization algorithm for dual resource constrained flexible job-shop scheduling problem. *International Journal of Production Research*, 54:1–13, 03 2016.