# A study on a Q-Learning algorithm application to a manufacturing assembly problem *

Miguel Neves, Miguel Vieira, Pedro Neto
University of Coimbra
Coimbra

April 18, 2023

## Abstract

The development of machine learning algorithms has been gathering relevance to address the increasing modelling complexity of manufacturing decision-making problems. Reinforcement learning is a methodology with great potential due to the reduced need for previous training data, i.e., the system learns along time with actual operation. This study focuses on the implementation of a reinforcement learning algorithm in an assembly problem of a given object, aiming to identify the effectiveness of the proposed approach in the optimisation of the assembly process time. A model-free Q-Learning algorithm is applied, considering the learning of a matrix of Q-values (Q-table) from the successive interactions with the environment to suggest an assembly sequence solution. This implementation explores three scenarios with increasing complexity so that the impact of the Q-Learning's parameters and rewards is assessed to improve the reinforcement learning agent performance. The optimisation approach achieved very promising results by learning the optimal assembly sequence 98.3% of the times.
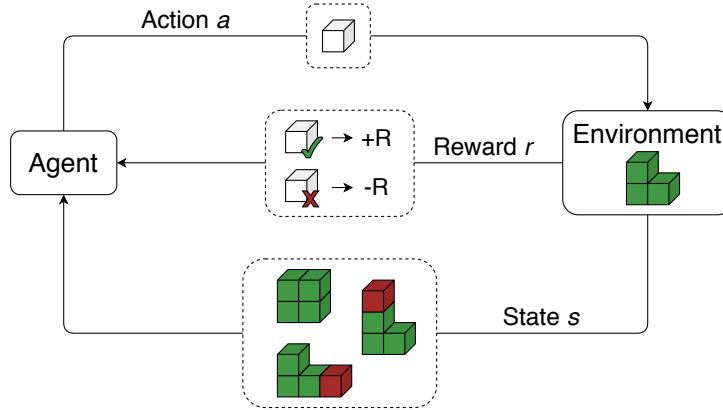
Figure 1: Agent and environment interaction.

Keywords: Reinforcement Learning, Q-Learning, Assembly Sequence, Optimisation

# 1 Introduction

Alongside the advent of product customization, industrial manufacturing processes are increasingly more complex, required to be highly flexible, resourceful, and efficient. Their related decision-support problems have been explored in literature with the development of traditional optimisation mathematical models (e.g. LP, IP, MILP or MINLP) and heuristic approaches [1], and where machine learning algorithms have been gathering particular relevance. However, most machine learning approaches rely on trial and error interactions that requires available data (e.g. supervised learning), which is often unavailable. A current approach to such optimisation problems is based on reinforcement learning (RL).

RL is a machine learning paradigm originally inspired by the way biological systems learn [2], where an agent (e.g. a human, a robot, a vehicle) interacts with the environment by taking actions, Fig. 1. As a consequence of the action taken, the environment defines states and rewards. A state is essentially a description of the situation of the environment and the rewards are an abstract concept that describes the feedback by which the success or failure of the agent is measured. This kind of learning is gathering relevance in problems that require decision making, by dynamically exploring a

solution that maximises the total rewards.

The main advantages of RL compared to other machine learning approaches are the fewer data available in the learning phase, which are the current states and expected rewards in a continuous learning process, fuelled by the interaction between the agent and the environment. Given its characteristics, RL methods are used in complex problems where there appears to be no obvious or easily programmable solution, such as game playing, robotics, control problems or operational research, and in problems where there is not enough labelled data such as anomaly detection problems. Since most of programming tasks are tedious and require years of expertise, RL algorithms can be applied to replace it with an intuitive process comprehensible even by an unskilled user.

## 1.1 Reinforcement learning applications

One of the most successful applications of RL in recent years was the development of the Go playing program known as AlphaGo. This program achieved a 99.8% winning rate against other Go programs and defeated the European Go champion by 5 games to 0 [3]. This program was trained by supervised learning from human expert moves and by reinforcement learning from self-play. This application was further improved (AlphaGo Lee) and was capable of defeating the 9 dan player Lee Sedol, winner of 18 international titles, 4 games out of 5 [4].

Also, in the field of game playing, RL was employed in a set of 49 Atari games by developing the DQN algorithm. This new algorithm was able to outperform the best RL methods at the time in 43 out of the set of 49 games. Furthermore, DQN performed at a level comparable to a professional player, achieving more than 75% of the human score in 29 out of the set of 49 games tested [5].

In recent years, the research on the applicability of RL is also increasing in the fields of decision-making and system control problems. By applying Q-Learning in a stock optimisation problem were achieved results up to 25% better when compared to traditional stock management algorithms, [6]. Wang and Usher studied the implementation of the Q-Learning algorithm for the usage of job agents when establishing routing decisions in a job shop environment, [7]. The authors discussed the effects of the Q-Learning application with the guidelines for future applications and recommendations for factor settings. Also, in dynamic job shop scheduling problem (DJSS) [8] pro-

posed the usage of RL with a Q-factor algorithm to improve the scheduling method's performance while considering random job arrivals and machine breakdowns. In a simulated environment, this proposed method achieved high performances. In automotive paint shops, colour changeovers between consecutive production orders are a source of costs due to the process of cleaning painting robots. In order to minimize these costs, production orders can be re-sequenced and grouped in identical colours as a batch. Leng et. al. propose the usage of the Deep Q-Network algorithm to solve this Colour-batching Re-sequencing Problem [9]. Huang et al. proved, through a simulation study, the effectiveness of the usage of Q-Learning in a maintenance problem where random failures of machines are highly disruptive [10]. The usage of reinforcement learning was also proposed to tackle the profit optimization problem of a single product production affected by deterioration failures, requiring both maintenance and repairs [11]. The performance in manufacturing work cells that utilise gantries to load and unload the materials and parts needed is highly dependent on the gantry movements in real operation. Ou et al. formulated the gantry scheduling problem as a RL problem through the usage of Q-Learning and demonstrated, from the simulation results, the capability of effectively reducing system production losses in real-time operations, [12]. In a later article, five different reward functions were devised based on different assumptions of the system. It was shown that the policy derived from systematic analyses of production loss significantly outperformed the other policies. Therefore, it was concluded that the level of understanding of the system and how this understanding is transmitted to the reward function greatly impacts the learning model's success [13]. In the areas of production manufacturing [14] proposed the usage of RL to improve assembly efficiency by a dual-arm robot and achieved higher performance when compared with other methods. In recent years, personalized production has emerged due to the increasing customer demand for more personalized products. This type of production, when compared with traditional production, has more uncertainty and variability. Such problems can be tackled by the usage of multi agent systems and reinforcement learning in a smart manufacturing environment. This approach was shown to be competitive in a dynamic environment [15]. The automation of shoemaking production lines is extremely challenging also due to the versatility of manual product customization. To tackle this problem a cyber-physical system artificial intelligence architecture was devised for the complete manufacturing of soft fabric shoe tongues by using Deep-Q reinforcement learning as a

4

means of achieving better control over the manufacturing process and convolutional and long short-term memory artificial neural network to enhance action speed [16].

In a collaborative assembly process context, [17] proposed an approach based on Interactive Reinforcement Learning to reduce the programming effort required by an expert. The learning approach is made in two steps. The first one consists of modelling simple tasks that constitute the assembly process, using task-based formalism. These modelled simple tasks are then used by the robotic system by proposing to the user a set of possible actions at each step of the assembly process via a graphic user interface (GUI). After the user selects the action, the robot performs it, progressing the assembly process while learning the assembly order. The framework also allows different users to teach different assembly processes to the robot. This proposed approach is based on Q-Learning and IRL and was successfully applied in a UR10 robot in an assembly process comprising tasks such as, picking, holding, mounting and receiving objects. Human-robot interactions have become abundant due to the increase of autonomous robotic operators. The consequence of such interaction is the introduction of a source of uncertainty and unpredictability from the human operator. Oliff et. al. presents a methodology for the implementation of a RL-based intelligent agent which allows a change in the robotic operator's behavioural policy in response to performance variation in the human operators [18]. Lastly, there has been an increase in the research of using digital twins in smart manufacturing environments. Xia et. al. developed a control methodology named Digital Engine capable of acquiring process knowledge, scheduling manufacturing tasks, identifying optimal actions and demonstrating control robustness [19]. Hu et. al. proposed a new graphic convolution layer named Petri-net convolution layer (PNC). When utilizing DQN with a PNC network better solutions are obtained for dynamic scheduling problems [20].

## 1.2   Main challenges and objectives

Even though many advances were made in recent years, some major challenges where noted by [17] when applying RL algorithms. The first challenge arises in the learning phase where the agent must do a trade-off between doing actions known to be effective (exploitation) and actions not yet explored (exploration), which is known as the exploration-exploitation dilemma [2]. Another common challenge is the "curse of dimensionality" [18], where to

5

ensure global optimality, data must be collected throughout the entire state-space, often infeasible in high-dimensional state-action spaces. The restrictions of expensive hardware, component's wear with economical and logistical consequences in the agent interactions with the physical world are known as the "curse of real-world samples". To reduce real-world interactions, a model can be used as a simulation system and the knowledge transferred to a real scenario. However, creating accurate models is very challenging and sometimes even impracticable. Small errors due to under-modelling can accumulate and make the simulation diverge from the real behaviour, which is known as "curse of under-modelling and model uncertainty". Finally, the desired behaviour in RL is often specified by the reward function, which is frequently easier than defining the behaviour itself, however, in practice, in some problems it may be astonishingly difficult. This RL challenge is often known as "curse of goal specification".

In this RL application, our approach aims to study the exploration-exploitation dilemma and the curse of goal specification, in which the type of agent properties (e.g. learning rate or reward signal) can influence the results obtained, here tackled by the assessment of the algorithm's learning parameters in the solution's outcome. In addition, this intends to support work planning with the demonstration of the RL method application regarding a feasible and efficient assembly sequence of a product. Therefore, the problem application resumes the optimisation of an assembly problem for a given object considering a time efficient solution.

The remainder of the paper is structured as follows: in Sect. 2, the problem formulation and modelling theory of the Q-learning algorithm is outlined; in Sect. 3, the case study is presented with the discussion of the results implementation for a set of assembly scenarios; and finalising, in Sect. 4, with the conclusions and future work.

## 2 Model formulation

### 2.1 Problem description

Due to the market diversification of on-demand product attributes, current production systems are being required to deal with assembly flexibility, in which one or multiple resources is in charge of all steps of product customisation. The tasks sequencing represents one of the major components directly

related to the efficiency of the assembly process. However, the assembly of a product is constrained by the product design and quality considerations, which often derives many possibilities in assembly sequence steps.

The problem considers a representation of an assembly job of a complex product containing different parts and tools, decomposed in a number of tasks which can be assembled in different sequences by an assigned resource. Along with the achievement of a feasible assembly process, improving the time efficiency is nontrivial due to the complexity of sequences' immediate/general time dependence of previous completed tasks. In order to cope with these issues, the goal of this work is to assess the effectiveness of an RL agent algorithm in the optimisation of task sequences based on real-time system states.

## 2.2   Mathematical approach

To provide a baseline comparison, the problem is formulated as a benchmark mathematical optimisation MILP model, given the following mathematical notations:

**Sets:**
$i$, $i'$, $i''$     Tasks

$k$, $k'$     Sequence step

**Subsets:**
$P_{ii'}$     General precedence tasks feasibility
$Q_{ii'}$     Immediate forbidden sequence of tasks

**Parameters:**
$\tau_{ik}$     Average processing time of task $i$ to sequence step $k$
$\Delta_{ii'}$     Processing time variation of task $i'$ given the previous completed task $i$

**Variables:**
$Y_{ik}$     Binary variable to assign task $i$ to sequence step $k$
$C_{ik}$     Completion time of task $i$ at each sequence step $k$
$C_{max}$   Makespan

The problem is formulated as an assignment problem of a set of tasks $i \in I$ to a set of sequence steps $k \in K$ based on the concept of a general precedence model, setting in Eq. 1 the objective function as the makespan minimisation.

The variable $C_{ik}$ defines the completion time of task $i$ at each step $k$, given the assignment of task $i$ to sequence step $k$ set by variable $Y_{ik}$. Considering that the last step $k = K^{last}$ comprises a feasible assembly combining all possible tasks, the total completion time of the optimal assembly sequence is minimised.

$$Minimise\ C_{max} \geq C_{ik} \quad \forall i, k = K^{last} \tag{1}$$

s.t.

$$\Sigma_i Y_{ik} = 1 \quad \forall k \tag{2}$$

$$\Sigma_k Y_{ik} = 1 \quad \forall i \tag{3}$$

$$Y_{ik} + Y_{i'k'} < 1 \quad \forall i, i' \in P_{ii'}, i \neq i', k < k', k \neq k' \tag{4}$$

$$C_{ik} \geq \tau_{ik} Y ik + (C_{i'k-1} + \Delta_{i'i} Y_{i'k-1}) \mid_{k>1} + $$
$$(\Sigma_{k'>k-1} \Sigma_{i'' \notin Q_{i''i}, i'' \neq i} \Delta_{i''i} Y_{i''k'}) \mid_{k>2} \forall i, i' \notin Q_{i'i,k} \tag{5}$$

$$C_{ik} \geq 0 \quad Y_{ik} \in \{0, 1\} \tag{6}$$

Regarding the formulation, Eqs. 2-3 considers the allocation constraints to allocate one and only one task to one step, and Eq. 4 guarantees the general precedence rule of two tasks (under subset $P_{ii'}$). Eq. 5 defines the completion time for every task and according to production step, ensuring the timing between two consecutive stages, defined by the allowed assembly sequence given by $Q_{i'i}$. Besides the processing time of each task per stage $\tau_{ik}$, the model considers the variation of time duration according to previous completed tasks, given by $\Delta_{ii'}$. Finally, Eq. 6 reassures the non-negativity and integrality of variables.

## 2.3 RL modelling theory

Given the anatomy of RL algorithms, they can be classified into three categories of methods which are value-function methods, policy search methods and actor-critic methods. The value-function methods, also known as critic-only methods, are based on the idea of initially discovering the optimal value function by fitting a value-function or a Q-function and then deriving the optimal policy from this. On the other hand, the policy search methods, also known as actor-only methods, search directly in the policy space by summing the rewards of sample trajectories, which is only possible if the search space is restricted. In the particular case of policy search algorithms, known as
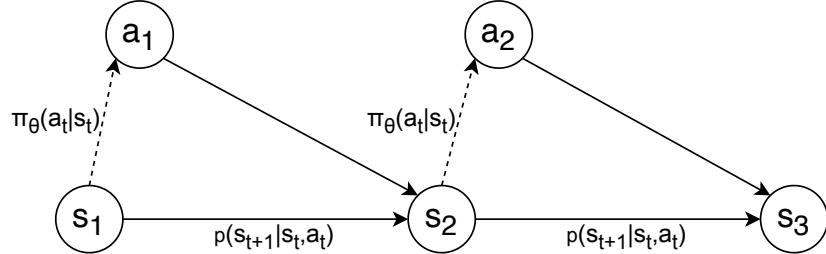
Figure 2: Markov decision process (MDP).

policy gradient methods, one step of gradient ascent is applied to the expected reward objective. Lastly, the actor-critic methods are a combination of both where the critic's function is to monitor the agent's performance by fitting a value function or a Q-function to determine when the policy must be changed by the actor. Moreover, the models can also be divided into model-free algorithms, which do not use models of the environment and are explicitly trial-and-error learners, and model-based algorithms, which use the model for planning or policy improvement.

RL algorithms are commonly formalised as Markov decision processes, where the agent observes the current state and decides the next suitable action. The reason for this formalism is the increased difficulty of computation by considering all the states and actions taken from the initial state. Using MDPs, the system only needs to keep track of the last state and action. However, it is important to understand that this Markov assumption leads to the loss of data, which in some situations might be relevant since rewards may be infrequent and delayed.

To define a Markov decision process, $M = \{S, A, T, r\}$, it is required to define a state space $S$, an action space $A$, a transition operator $T$ and a reward function $r$. The state space is a set of valid states $s$ the system can occupy, $s \in S$. The action space is the set of possible actions $a$ the agent can take such that $a \in A$. In Markov decision processes the transition probabilities are not only conditioned on the previous state but also on the previous action, $p(s_{t+1} \mid s_t, a_t)$. Since the policy is the mapping of the states to actions, $\pi_\theta(a_t \mid s_t)$, a graphical representation of a Markov decision process can be observed in Fig. 2.

The learning agent must evaluate whether if it is being successful in the task. An agent can discern good from bad events based on the reward signal, which is analogous to the way the humans learn when experiencing pain or

pleasure. This is the primary source of improvement of the policy since if the action selected in a certain state returns a low reward, then the policy may be changed so that, when faced by the same exact state, the policy selects a different and more rewarding action. However, the agent's goal is to maximise the accumulated reward over time through the actions chosen. However, an action with a high immediate reward might not be the optimal choice since it may lead to a lower accumulated reward over the future. To tackle this issue, there are two important concepts, the value function and the quality function, usually known as Q-function. Given a policy, the value function is defined as the total expected reward from a given state $s_t$:

$$V^\pi(s_t) = \sum_{t'=t}^{T} E_{\pi_\theta}[r(s_{t'}, a_{t'}) \mid s_t] \qquad (7)$$

The Q-function is, on the other hand, the total expected reward from taking the pair action $a_t$ in the state $s_t$:

$$Q^\pi(s_t, a_t) = \sum_{t'=t}^{T} E_{\pi_\theta}[r(s_{t'}, a_{t'}) \mid s_t, a_t] \qquad (8)$$

Actions must therefore be selected based on value judgements because the agent's goal is to maximise the accumulated reward over time. Unfortunately, while rewards are given directly by the environment, values must be estimated multiple times within the sequence of observations.

### 2.3.1   Q-Learning

In order to implement the algorithm, it is necessary to understand the Q-Learning parameters used, such as how the Q-table is updated. The value iteration update is done at each step through the Bellman Equation, which consists on the weighted average of the old Q-value and the new information obtained, where $\alpha$ corresponds to the learning rate, $\gamma$ to the discount factor, $r_t$ to the received reward when moving from state $s_t$ to $s_{t+1}$, $Q^{new}(s_t, a_t)$ to the new Q-value of the state $s_t$ and action $a_t$, $Q(s_t, a_t)$ to the old Q-value of the state $s_t$ and action $a_t$ and $\max_a Q(s_{t+1}, a)$ to the estimate of the optimal future Q-value:

$$Q^{new}(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha \times [r_t + \gamma \times \max_a Q(s_{t+1}, a) - Q(s_t, a_t)] \qquad (9)$$

The learning rate parameter has values between 0 and 1 and influences the extent on how the new information changes the current state value, which means that a lower learning rate leads to a longer learning time. However, it is important to note that a higher learning rate may lead to suboptimal results or even divergence in non-deterministic scenarios. The discount factor determines the importance of future rewards, so the lower its value the less meaningful are the future rewards. If the discount factor has the value 0, only the current reward is considered. The selection of the action is made using an epsilon greedy search, i.e. the agent selects a random action with probability $\varepsilon$ and otherwise selects the action greedily, with probability $1 - \varepsilon$, by selecting the action with the highest Q-value. The value of epsilon ($\varepsilon$) decays based on a decay rate known as epsilon decay. The algorithm can be summarized as follows:

---

**Algorithm** Q-Learning

---

01:    $Q(s, a)$ initialised arbitrarily
02:    **For each** episode $i$ **do**
03:      state $s$ initialised
04:      **For each** step $j$ **do**
05:       action $a$ chosen from state $s$ using policy derived by Q (e.g. $\varepsilon$-greedy)
06:       action $a$ taken
07:       reward $r$ and state s' observed
08:       $Q(s, a) = Q(s, a) + \alpha \times [r + \gamma \times \max_{a'} Q(s', a') - Q(s, a)]$
09:       $s = s'$
10:      **end for**
11:    **end for**

---

# 3    Experiments and results

## 3.1    Case study

The object chosen for the assembly problem is an aeroplane toy from the Yale-CMU-Berkeley Object and Benchmark Dataset [19, 20] (Fig. 3), which process is studied and optimised through the implementation of a RL methodology.
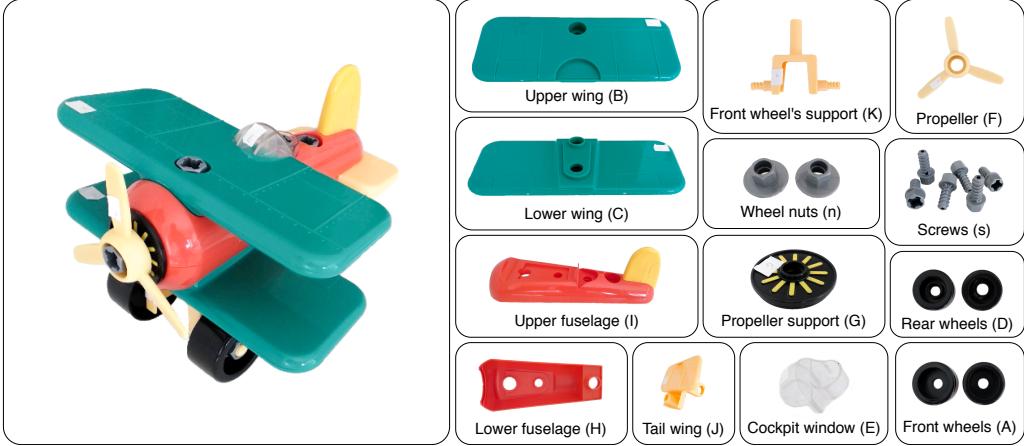
Figure 3: Aeroplane from the Yale-CMU-Berkeley Object and Benchmark Dataset.

As follow, the object components and assembly structure are thoroughly analysed. As a preliminary evaluation, the problem solution is derived from the mathematical optimisation model. Then, the assembly problem is formulated using Q-Learning and is implemented in two different scenarios where the agent must learn a feasible assembly sequence, i.e. an assembly sequence that respects all the task precedencies. The scenarios consider the distinct time durations of each tasks incorporated in the decision-making algorithm to foment the learning of an optimised time efficient assembly sequence. The algorithm's learning parameters are individually analysed in order to improve the agent's performance.

## 3.2 Assembly analysis

The aeroplane object is comprised of 9 structural parts and 2 types of fasteners, which are displayed in Table 1 and Table 2.

After subdividing the aeroplane into parts and fasteners, the assembly process was subdivided in a total of 8 tasks. In the Table 3 each task is associated with the corresponding parts and fastener required. It is important to note that some parts can be used in more than one task. For the assembly to be complete, every task must be executed without repetitions, therefore the number of different assembly sequences is $n! = 8! = 40320$.

However, the feasible number of assembly sequences is lower than the

Table 1: Aeroplane's parts.

| Part | Aeroplane part's description | Number of parts |
|---|---|---|
| A | Front wheels | 2 |
| B | Upper wing | 1 |
| C | Lower wing | 1 |
| D | Rear wheels | 2 |
| E | Cockpit window | 1 |
| F | Propeller | 1 |
| G | Propeller's support (engine) | 1 |
| H | Lower body of the aeroplane (lower fuselage) | 1 |
| I | Upper body of the aeroplane (upper fuselage) | 1 |
| J | Rear body of the aeroplane (tail wing) | 1 |
| K | Front wheel's support | 1 |

Table 2: Aeroplane's fasteners.

| Fastener | Fastener's head type | Number of fasteners |
|---|---|---|
| n | Wheel nuts | 2 |
| s | Screws | 5 |

Table 3: Parts and fasteners associated to each task and their respective quantities.

| Tasks | Parts | | | | | | | | | | | Fasteners | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | A | B | C | D | E | F | G | H | I | J | K | n | s |
| 1 | | | | | | | 1 | 1 | 1 | | | | |
| 2 | | | | | | | | | | 1 | | 1 | |
| 3 | | | | | | 1 | | | | | | 1 | |
| 4 | | | 1 | | | | | | | | | 1 | |
| 5 | | 1 | | | | | | | | | 1 | 1 | |
| 6 | | 1 | | | 1 | | | | | | | 1 | |
| 7 | 1 | | | | | | | | | | 1 | | 2 |
| 8 | | | | 1 | | | | | | 1 | | 1 | |

Table 4: Precedence task's feasibility, $P_{ii'}$.

| Task | Precedence task |
|------|-----------------|
| 1 | None |
| 2 | 1 |
| 3 | 1 |
| 4 | 1 |
| 5 | 1 and 4 |
| 6 | 1 |
| 7 | None |
| 8 | None |

previously calculated one, due to the fact that certain tasks require other tasks to be previously completed. Such precedence sequence dependencies are displayed in the Table 4. When taking into account the task dependencies, the feasible number of assembly sequences is 3360, which corresponds to only 8.3% of all assembly sequences.

In resume, the assembly process is subdivided in 8 different tasks, or actions, and the assembly can be considered as complete when all the 8 tasks have been executed. Therefore, the MDP's states can be defined by the updated assembly status at each task, where the initial state would correspond to the situation where none of the actions was executed, the final state when all the actions were completed and the aeroplane is assembled, represented by the binary number indicating whether the task has been executed. The digit in the binary number corresponds to the task number. If the task $n$ has been executed, 1 is set to the $n^{th}$ digit in the binary number. Otherwise, the $n^{th}$ digit is set as 0 (Fig. 4). The initial state would then be represented as 00000000 and the final state would correspond to 11111111, which in decimal notation corresponds to 0 and 255 respectively, thus, the number of states is 256. Similarly, due to task precedences, there are only 100 possible states.

Finally, for this case, the tasks' average time were estimated $\tau_{ik}$ (Table 5) as well as the increase/decrease variances on the average times with respect to the tasks previously done $\Delta_{ii'}$ (Table 6). In this table is also shown shaded cells in grey with the immediate forbidden sequences, which add as restrictions to the previous general precedence.
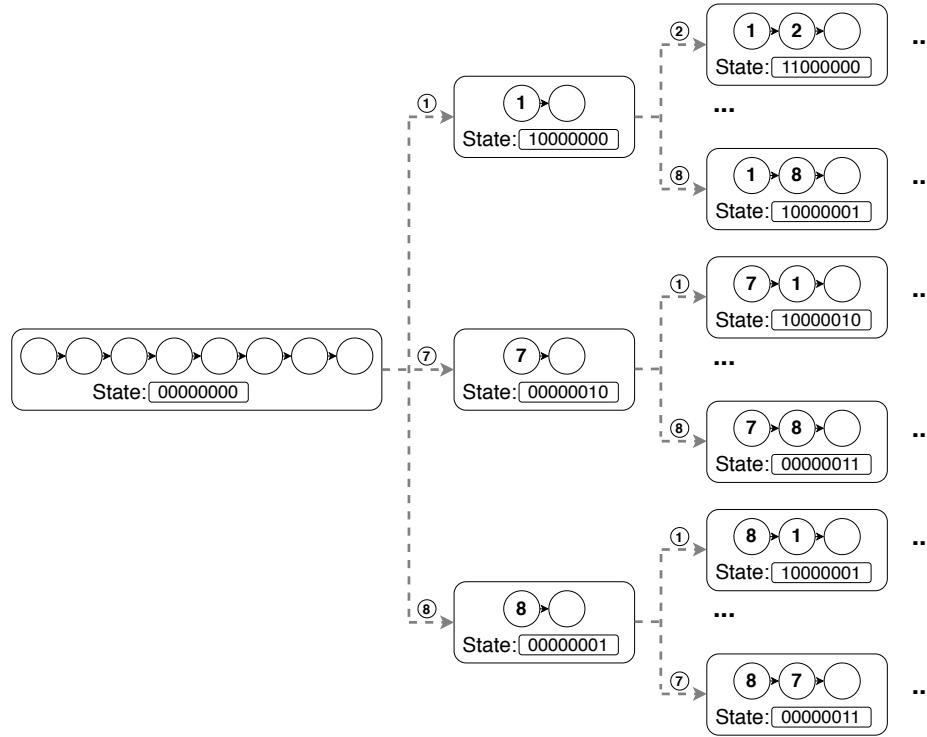
Figure 4: MDP's states and actions scheme.

Table 5: Task's average time, $\tau_{ik}$.

| Task | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|---|
| Average time [time units, t.u.] | 10 | 7 | 8 | 6 | 12 | 8 | 11 | 9 |

15

Table 6: Tasks' variation in respect to the average time given completed tasks $\Delta_{ii'}$ [t.u.] and immediate forbidden sequences, $Q_{ii'}$.

| Task | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|------|---|---|---|---|---|---|---|---|
| Task 1 done | | | | | | | 0 | 0 |
| Task 2 done | | | -1 | -1.5 | 0 | -1 | 0 | 1 |
| Task 3 done | | 0 | | 0 | 0 | 0 | 0 | 0 |
| Task 4 done | | -0.5 | 0 | | | 0 | 0 | 0 |
| Task 5 done | | -1 | -0.5 | | | -2 | 1 | 0 |
| Task 6 done | | 0 | 0 | 0 | 0 | | 0 | 0 |
| Task 7 done | 0 | 0 | 0 | 0 | 0 | 0 | | 0 |
| Task 8 done | 0 | 0 | 0 | 0 | 0 | 0 | 0 | |

Table 7: Mathematical model statistics

| Model output | #Total variables | #Binary variables | #Equations | Optimality gap | CPU (s) | Solution |
|---|---|---|---|---|---|---|
| | 129 | 64 | 550 | 0.0 | 0.34 | 65.0 |

## 3.3 Mathematical optimisation solution

To previously assess a baseline solution for the comparison of the Q-learning algorithm performance, the mathematical optimisation model defined in section 2.2 is implemented in GAMS (29.1.1 ver.), using CPLEX (12.8.0.0 ver.) solver, run in an Intel(R) Core(TM) i7-7700HQ @2.80GHz with 16GB RAM. Given the problem description and data defined in section 3.2, the optimal solution obtains an assembly task sequence $1 \to 8 \to 4 \to 7 \to 5 \to 2 \to 6 \to 3$ with the value of 65 time units (t.u.). The solution complies with the given feasibility constraints for an optimality gap of 0 in under 1 second. The model statistics are presented in Table 7.

## 3.4 Q-learning algorithm - Scenario I: Learning an assembly sequence based on estimated task average times and variances

To better understand the scenario's results of the Q-learning implementation using Matlab R2020a, it is important to introduce the concept of experiment.

An experiment comprises the algorithm's learning phase and the result obtained at the end. The learning phase in an experiment takes place over various episodes dictated by the maximum number of episodes per experiment. In this specific case, an episode starts with no tasks done and ends when all tasks have been successfully completed or when the maximum number of steps has been reached. However, in all scenarios the algorithm considers that, whenever the Q-Learning agent selects an impossible action, the current state does not change, which means that the sequence is penalised for requiring more than 8 steps to complete an episode.

At the end of the experiment, the agent selects the actions based solely on the Q-Values, which means that after learning, the agent always selects the same assembly sequence (expressed as learned assembly sequence or experiment's result). In order to study statistically relevant solutions, each set of parameters and rewards is replicated in 120 experiments.

With the objective of understanding how well the agent would be able to compare the efficiency of the feasible assembly sequences, the rewards are assigned to the processing time of each task. These were then defined through (Eq. 10), where $R(s,a)$ is the reward for taking the action $a$ in the state $s$, $r_m$ is the reward multiplier, $r_s$ the reward shift, $r_p$ the reward penalty and $T(s,a)$ the predefined time it takes to complete the action $a$ in the state $s$, that is calculated by summing the respective variances to the task's average time:

$$\begin{cases} R(s,a) = r_m \times (-T(s,a) + r_s) & if\ possible\ action \\ R(s,a) = r_p & if\ impossible\ action \end{cases} \quad (10)$$

Since the RL algorithm's goal is to maximise the accumulated reward, in order to learn the most time efficient assembly sequence (minimise the assembly sequence time), the value matrix $T(s,a)$ must be subtracted. The $r_s$ shifts each reward by its value and, as a result, shifts the accumulated reward by eight times its value. The $r_m$, on the other hand, multiplies the shifted reward. Consequently, the accumulated reward is also multiplied by $r_m$. If $r_m$ and $r_s$ have the values of 1 and 0 respectively, the accumulated reward is equal, in absolute values, to the duration of the assembly sequence. The accumulated rewards for all feasible assembly sequences for these exact values of $r_m$ and $r_s$ are displayed in the Fig. 5.

As it can be observed in Figure 5 (B), the most common accumulated reward, corresponding to 18.4% in all feasible assembly sequences, is -69.5, which correlates to the corresponding total assembly time. An example of
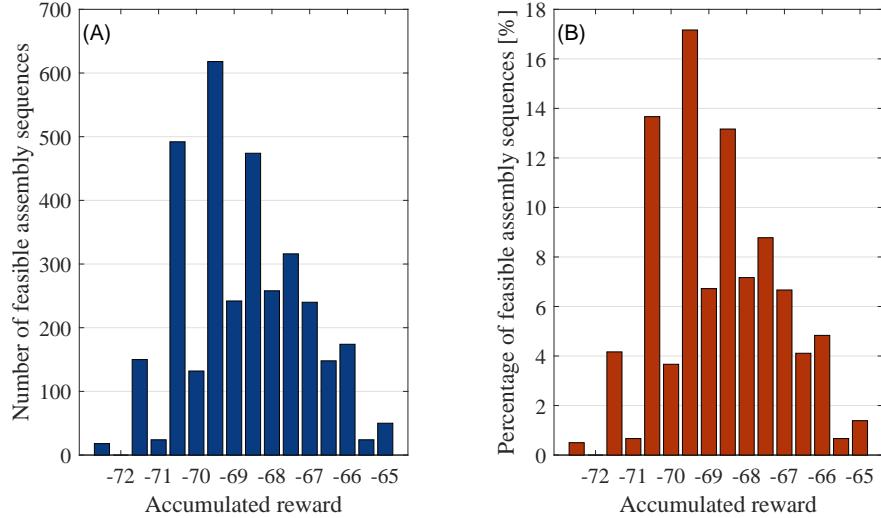
Figure 5: Distribution by number (A) and percentage (B) of feasible assembly sequences' accumulated rewards.

such an assembly sequence is $8 \rightarrow 1 \rightarrow 3 \rightarrow 4 \rightarrow 7 \rightarrow 2 \rightarrow 6 \rightarrow 5$ where the accumulated reward is $\Sigma R(s,a) = -9 - 10 - 8 - 6 - 11 - 6.5 - 7 - 12 = -69.5$. It is also possible to observe in the Fig. 5 (A) that there are 50 feasible assembly sequences with the maximum accumulated reward of -65 (optimal accumulated reward solution as shown on the mathematical optimization model).

In an initial sensitivity analysis, the rewards ($r_s$ and $r_p$) were tested individually (Fig. 6) with the parameters in the Table 6 and with the values 0, 1, and -10000 for the $r_s$, $r_m$ and $r_p$ respectively. The comparison of the performances for the various sets of parameters and rewards considers three indicators: the mean accumulated reward, normalised for a $r_m$ of 1 and a $r_s$ of 0; the percentage of times the agent learned one of the 50 optimal assembly sequences; and the percentage of times the agent failed to learn one feasible assembly sequence in the 120 experiments. When the agent failed to learn a feasible assembly sequence, the number of experiments was increased so that the mean would reflect 120 correctly learned assembly sequences. The reason for this rule was the high penalty on the mean of an incorrectly learned assembly, which would turn unfeasible a correct comparison between sets.

When observing the Fig. 6 (A), it can be concluded that a negative $r_s$ value increases the likelihood of incorrectly learning an impossible assembly
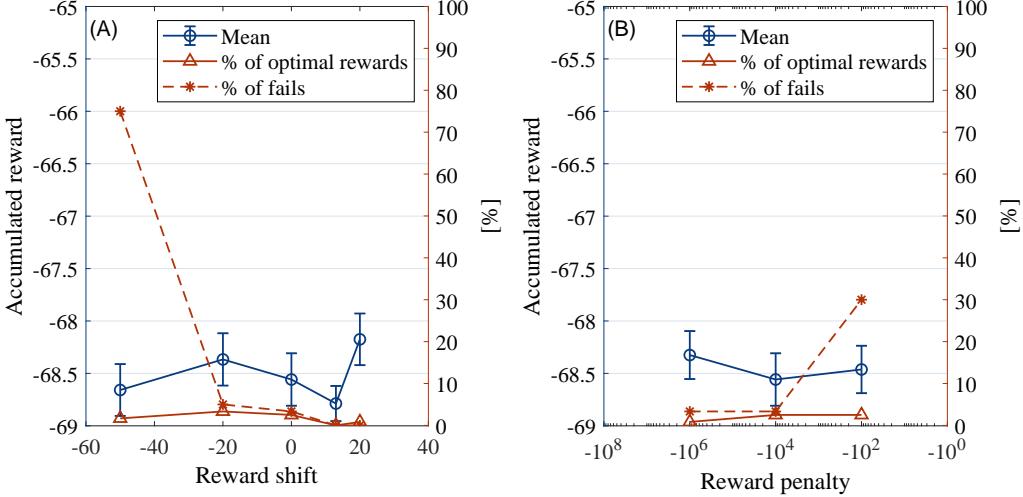
Figure 6: Impact of the reward shift (A) and reward penalty (B) on the agent's performance.

sequence because the agent is not able to differentiate between the penalties and the accumulated rewards shifted to values increasingly more negative. Also, a positive $r_s$ may lead to better results, as seen in the mean increase for the value of 20. The reward penalty, as seen in the Fig 6 (B), understandably, influences the percentage of fails since an action with a higher $r_p$ is more likely identified as an incorrect action, i.e. the bigger the penalty the lower the percentage of fails.

With the objective of understanding the impact of the $r_m$ and the maximum number of episodes, they were individually changed maintaining the parameters and rewards of the previous sensitivity analysis, except for the $r_s$ and $r_p$ with the new values of 20 and -1000000 respectively. Also, in the experiments where the maximum number of episodes was altered, the selected $r_m$ used was 5.

In terms of the $r_m$ sensitivity analysis displayed in the Fig. 7 (A), there are no statistically significant improvements in the agent's performance. However, the value chosen for $r_m$ in future sets of parameters and rewards is 20 as it accomplished the best results. In respect to the maximum number of episodes, presented in the Fig 7 (B), it is possible to conclude that increasing the maximum number of episodes per experiment leads to an increase in performance (both visible in the mean and in the percentage of optimal accu-
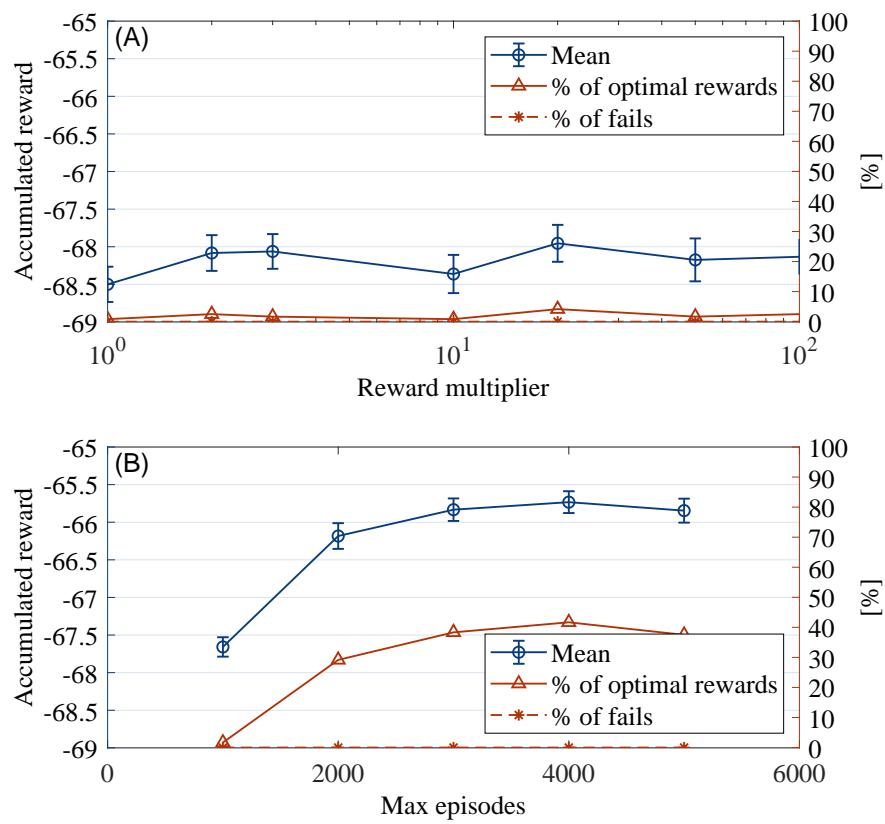
Figure 7: Impact of the reward multiplier (A) and maximum number of episodes per experiment (B) on the agent's performance.
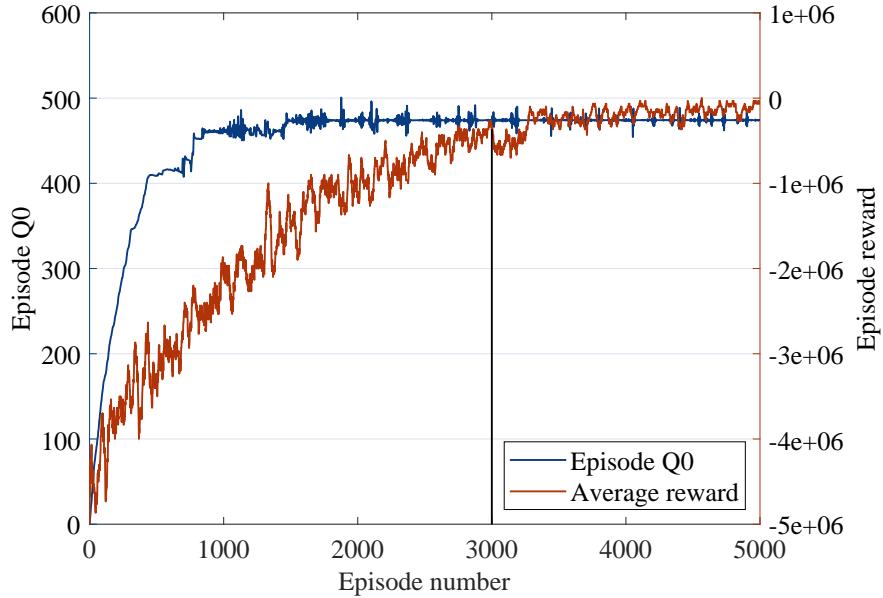
Figure 8: Evolution of the episode reward over the episodes for a value of epsilon decay of 0.0001.

mulated rewards) until a certain value in which it seems to plateau. Though, is important to remember that a higher value for the maximum number of episodes is related to the amount of times the experiment has to be repeated for the agent to learn. This means that it is essential to maintain the number as low as possible since in a real scenario it corresponds to the amount of assemblies required for the learning process. Thus, for these parameters, and specially for the epsilon decay of 0.0001, the optimal maximum number of episodes is 3000.

The optimal maximum number of episodes is dependent on the epsilon decay's value since a lower epsilon decay leads to a slower increase in the greedy selection by the agent and, as such, requires more episodes in the learning phase. In order to identify the relationship between these two parameters, a graph of the evolution of the episodic accumulated reward in one of the experiments (with 5000 maximum number of episodes) is analysed (Fig. 8).

When analysing the Fig. 8, it is possible to observe that after the identified optimal maximum number of episodes the episodic accumulated reward does not greatly increase. This could explain why increasing the maximum
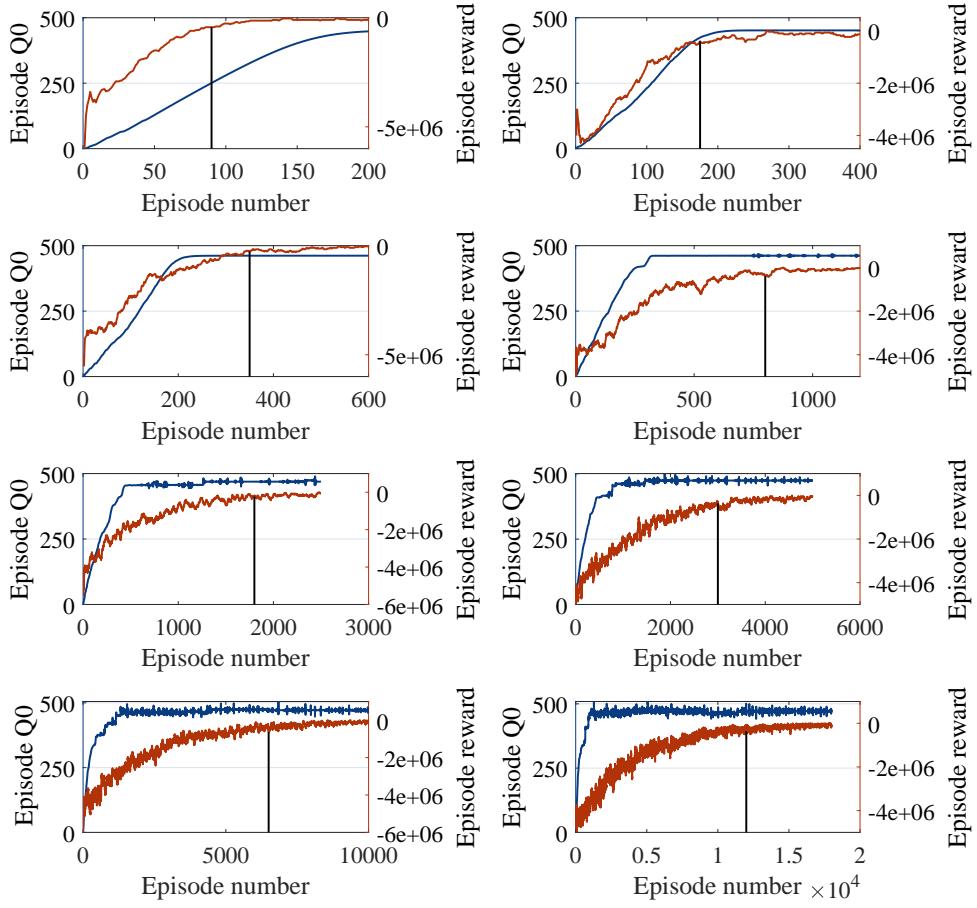
21

Figure 9: Evolution of the episode Q0 (blue line) and episode reward (orange line) over the episodes for various values of epsilon decay.

number of episodes further does not lead to a significant change in the results in the Fig. 7. Similar graphs were analysed for different values of epsilon decay in order to identify in which episode the episodic accumulated plateaus during an experiment to select this value as the optimal maximum number of episodes per experiment for the given epsilon decay (Fig. 9 and Table 8).

From the values available in the Table 8, a linear regression was devised using a logarithmic scale in both axes. As shown in the Fig. 10, the data approximated fits the function $y = 0.50528 \times x^{-0.95747}$ where $y$ is the maximum number of episodes and $x$ the epsilon decay.

Additional new experiments were run with the parameters shown in the

Table 8: Maximum number of episodes selected for each epsilon decay value.

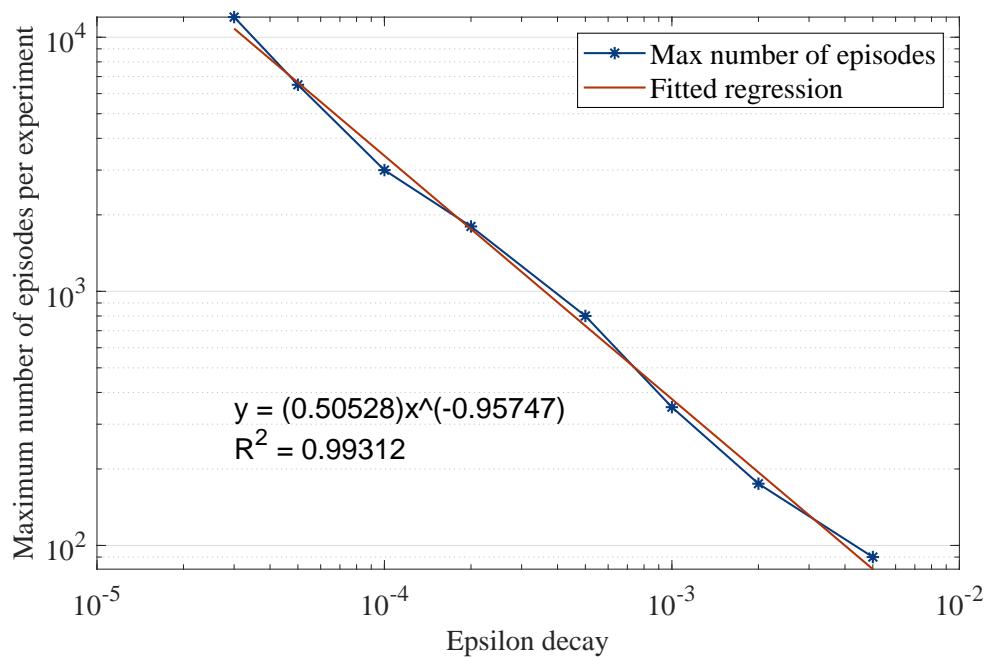| Epsilon decay | 0.005 | 0.002 | 0.001 | 0.0005 | 0.0002 | 0.0001 | 0.00005 | 0.00003 |
|---|---|---|---|---|---|---|---|---|
| Max episodes | 90 | 175 | 350 | 800 | 1800 | 3000 | 6500 | 12000 |



Figure 10: Graph of maximum number of episodes over epsilon decay.

23

Table 9: Parameters for the pair epsilon decay and maximum number of episodes experiment.

| Parameter | Value |
| --- | --- |
| Learning rate | 1 |
| Discount factor | 1 |
| Epsilon | 0.9 |
| Max steps per episode | 8 |
| Reward shift | 20 |
| Reward multiplier | 20 |
| Reward penalty | -1000000 |

Table 9, apart from the epsilon decay and maximum number of episodes which were based on the Table 8. Results are displayed in the Fig. 11.

As can be observed in the Fig. 11, the increase in the maximum number of episodes, accompanied by the respective decrease in the epsilon decay, leads to better results with an increase in the mean and in the percentage of optimal rewards. It is also possible to notice that such increase only starts around 1800 and 3000 maximum number of episodes.

## 3.5 Q-learning algorithm - Scenario II: Learning an assembly sequence based on measured task average times and estimated variances

In the second scenario, task time input data was measured and repeated 10 times (Table 10) so that the average processing times are now confirmed (approximation is used for simplification).

As in scenario I, the tasks' average times have variations in respect to the corresponding precedence tasks (Table 11). It is important to notice that the tasks variability complexity has increased from the scenario I, so that there would be a larger variety of accumulated rewards and a lower number of assembly sequences with the largest accumulated reward, i.e. optimal assembly sequences.

The variability was also increased by introducing the tool changeover time. Since there are two different types of fasteners, the fastening device's tool must be switched during the assembly process. Regarding the fastening
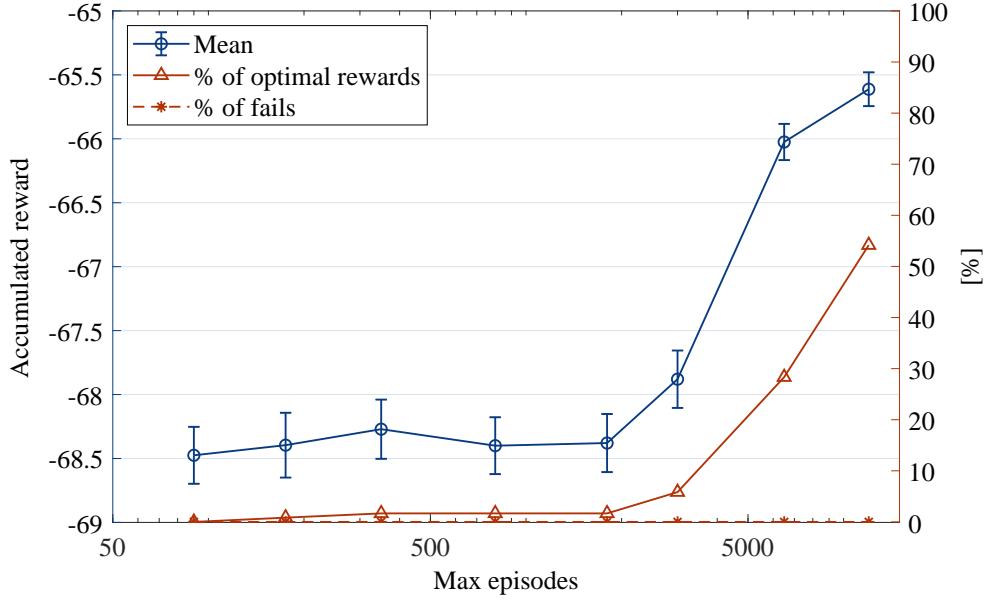
Figure 11: Impact of the pairs of epsilon decay and maximum number of episodes on the agent's performance.

Table 10: Task's time measurements.

| Task | | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|------|---|------|------|------|------|------|------|------|------|
| | 1 | 6.36 | 9.10 | 8.59 | 6.75 | 10.62 | 9.87 | 12.36 | 10.06 |
| | 2 | 6.28 | 9.15 | 10.30 | 8.22 | 10.52 | 11.31 | 12.39 | 8.50 |
| | 3 | 4.71 | 8.00 | 8.70 | 7.90 | 9.82 | 11.56 | 12.88 | 8.03 |
| Measured | 4 | 5.38 | 8.15 | 10.49 | 8.29 | 9.25 | 11.00 | 10.49 | 9.09 |
| tasks | 5 | 5.80 | 8.30 | 8.75 | 7.25 | 9.44 | 8.95 | 11.22 | 8.90 |
| [time units, | 6 | 6.71 | 8.52 | 8.97 | 7.40 | 10.04 | 12.16 | 11.04 | 9.37 |
| t.u.] | 7 | 5.73 | 8.17 | 9.00 | 7.95 | 9.07 | 9.24 | 11.94 | 7.70 |
| | 8 | 7.31 | 8.05 | 8.29 | 6.30 | 10.76 | 10.01 | 10.34 | 8.70 |
| | 9 | 5.16 | 7.62 | 8.33 | 7.75 | 9.95 | 9.56 | 10.14 | 8.15 |
| | 10 | 5.89 | 6.55 | 8.50 | 7.00 | 9.30 | 10.09 | 11.98 | 9.71 |
| Mean [t.u.] | | 5.93 | 8.16 | 8.99 | 7.48 | 9.88 | 10.38 | 11.48 | 8.82 |
| Average time | | 6 | 8 | 9 | 7.5 | 10 | 10.5 | 11.5 | 9 |

25

Table 11: Tasks' variation in respect to the average time given completed tasks $\Delta_{ii'}$ [t.u.] and immediate forbidden sequences, $Q_{ii'}$.

| Task | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|---|
| Task 1 done | | | | | | | 0 | 0 |
| Task 2 done | | | -2 | -3 | -0.5 | -2 | 0 | 1.5 |
| Task 3 done | | 0 | | 0 | 0 | 0 | 0 | 0 |
| Task 4 done | | -1 | 0 | | | -1.5 | 0 | 0 |
| Task 5 done | | -2 | -1 | | | -3 | 2 | 0 |
| Task 6 done | | -1 | -0.5 | 1 | -0.5 | | 0 | 0 |
| Task 7 done | 0 | 0 | 0 | 0 | 0 | 0 | | 0 |
| Task 8 done | 0 | 0 | 0 | 0 | 0 | 0 | 0 | |

device, there are two main assumptions made: the assembly process starts without any tool placed and the tool changeover lasts three time units to be performed. In order to introduce the tool changeover, the number of states must be increased. The states, apart from the binary number that define the completion of each task, can have three possible indexes (0, 1 and 2). The index 0 indicates that the fastening device does not have any tool placed (start of the assembly), the index 1 indicates that the fastening device has the screwdriver applied, and the index 2 indicates the nut driver is applied. The new total number of states is 511, since in the first state the fastening device has no tool and in any other state it can have either the first or the second tool. Due to task dependencies and tool selections, the number of possible states is 149.

With the respective changes to the average times and time variances the new distribution of accumulated rewards from the feasible assembly sequences can be observed in the Fig. 12.

The new distribution has, as previously stated, a larger variance of accumulated rewards and the highest accumulated reward or optimal accumulated reward is shared only by 2 assembly sequences (Fig. 12 (B)), which are $7 \to 1 \to 8 \to 2 \to 4 \to 5 \to 6 \to 3$ and $7 \to 8 \to 1 \to 2 \to 4 \to 5 \to 6 \to 3$, and has the value of -64. In this new scenario, it may be easier to understand the impact of the set's parameters on the agent's performance. The reward shift was modified for the values (0, 3, 6, 7, 8, 9, 10 12, 15) while using the values of the Table 9, apart from the epsilon decay and maximum number of episodes, which had the values of 0.00005 and 6500 respectively. The results
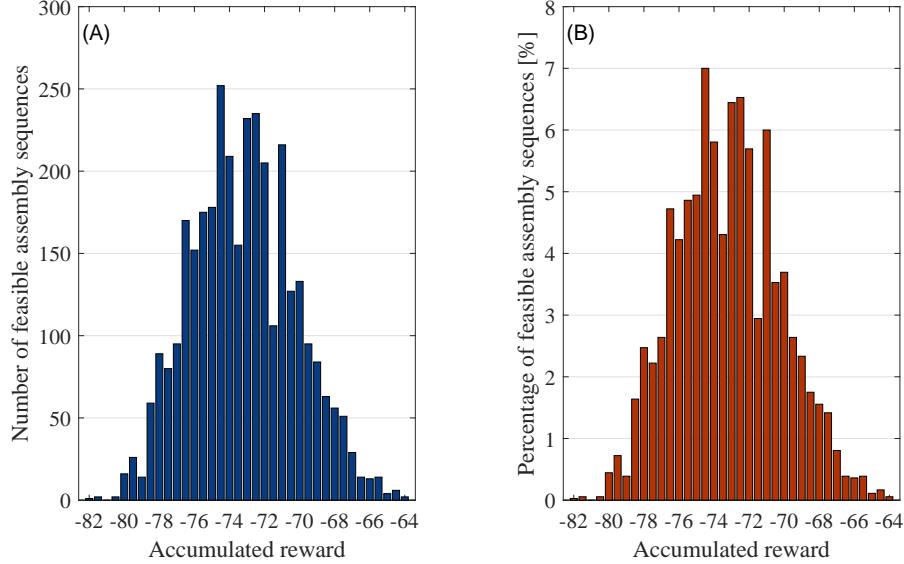
Figure 12: Distribution by number (A) and percentage (B) of feasible assembly sequences' accumulated rewards.

of the multiple sets of 120 experiments are displayed in the Fig. 13.

The mean of all the 37 possible values of accumulated reward is -73 (for a $r_s$ of 0 and $r_m$ of 1), and therefore, if subdivided evenly, each task would have a reward of -9.125, which we will define as mean task reward. A value of $r_s$ equal to the mean task reward shifts the accumulated rewards to a position where they are evenly separated into positive and negative. When analysing the Fig. 13 it is possible to identify that the best accumulated reward occurs for a value of the reward shift of 8. Also, it is important to notice that the percentage of fails decreases with the increase of the reward shift and is approximately 0 for values higher or equal to 9. Thus, the optimal reward shift may be related to the mean task reward, but it may be relevant to confirm this relationship with a different scenario. A value of the reward shift slightly lower than the mean task reward may improve the mean accumulated reward since a larger number of the accumulated rewards are negative. However, it also increases the percentage of fails, which is highly prejudicial for a real scenario. For that reason, the optimal value for the reward shift is 9. With this new $r_s$ value, the learning rate and the discount factor were individually analysed (Fig. 14).

Since in both cases the confidence interval is very small, even small dif-
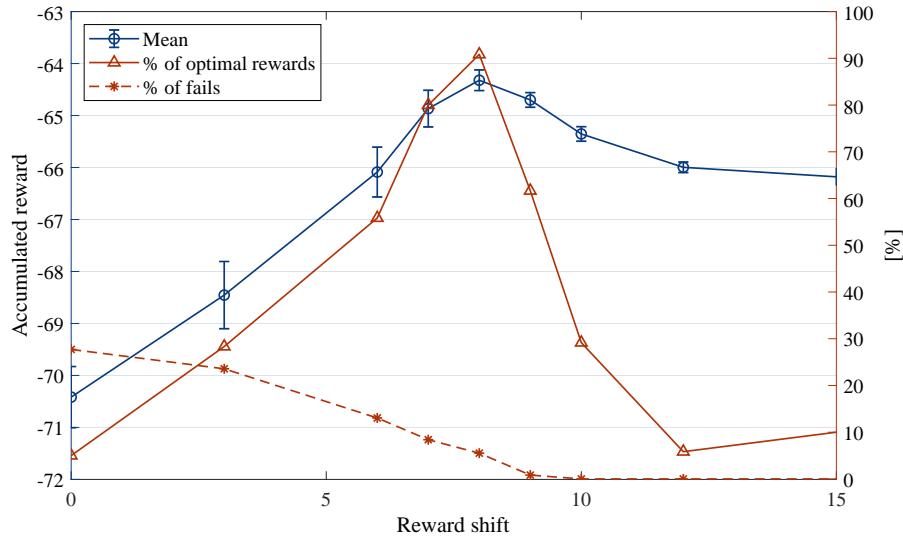
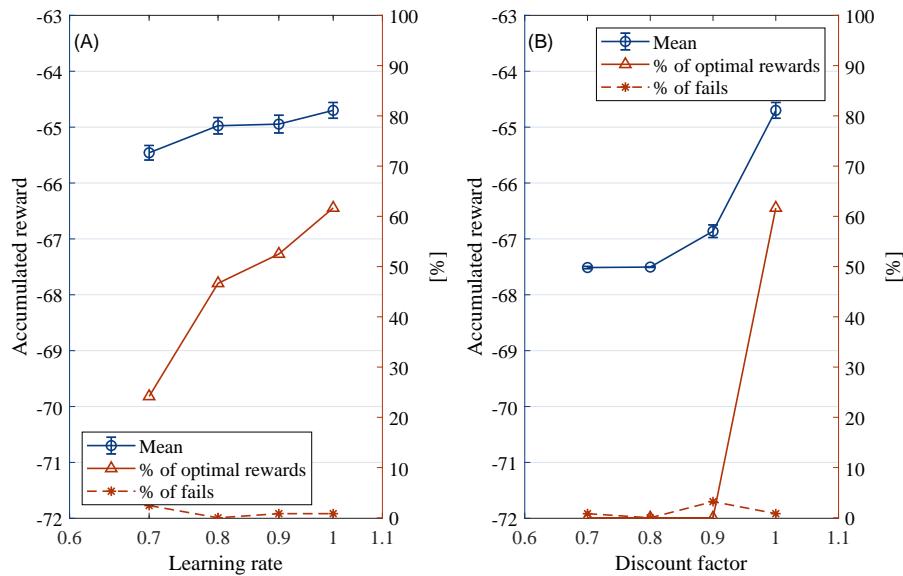Figure 13: Reward shift's impact on the agent's performance.



Figure 14: Learning rate's and discount factor's impact on the agent's performance.
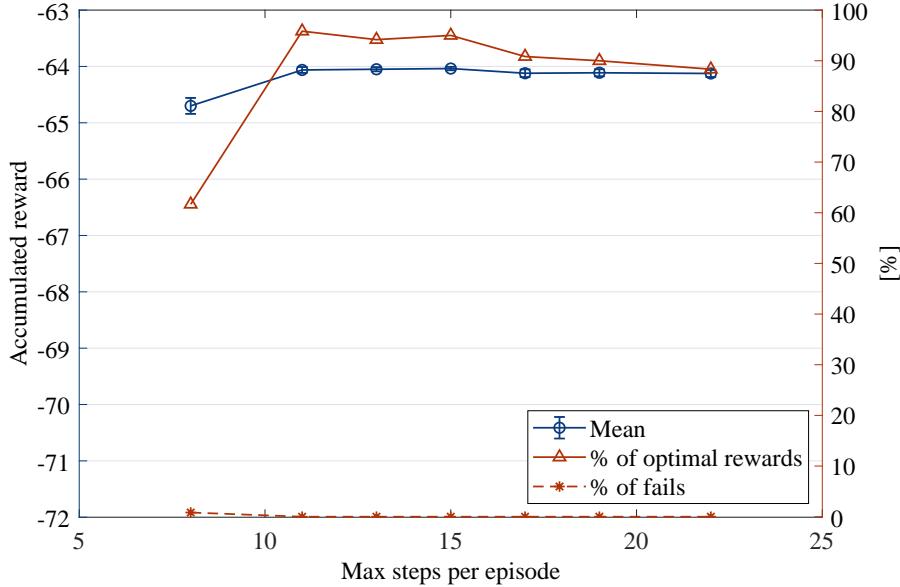
Figure 15: Maximum steps per episode's impact on the agent's performance.

ferences in the mean are significant. Both in the learning rate and in the discount factor, it can be concluded that the optimal value is 1, however, in the discount factor case the difference in the mean and in the percentage of optimal results is more accentuated. Then, with the same set's parameters as before, the maximum steps per episode were individually analysed (Fig. 15).

It is possible to conclude that an early increase in the maximum number of steps per episode leads to a significant increase both in the mean and in the percentage of optimal rewards. This shows that additional steps beyond the number of assembly tasks slightly contributes to improve the agent learning process, although further increase in this value does not significantly alter the results. The value 15 was selected for this parameter.

Lastly, with all the other parameters decided, the reward multiplier's impact was studied with various values (1, 5, 9, 11, 13, 15, 19, 25, 30) and the experiments results are visible in the Fig. 16.

It can be observed that there is an optimal value for $r_m$, since the increase in the value of the reward multiplier leads to a steady increase both in the mean and in the percentage of optimal rewards. For the smaller values of the reward multiplier, the percentage of fails is nonzero. Based on the graph, it can be defined that the optimal reward multiplier value is 13. To confirm
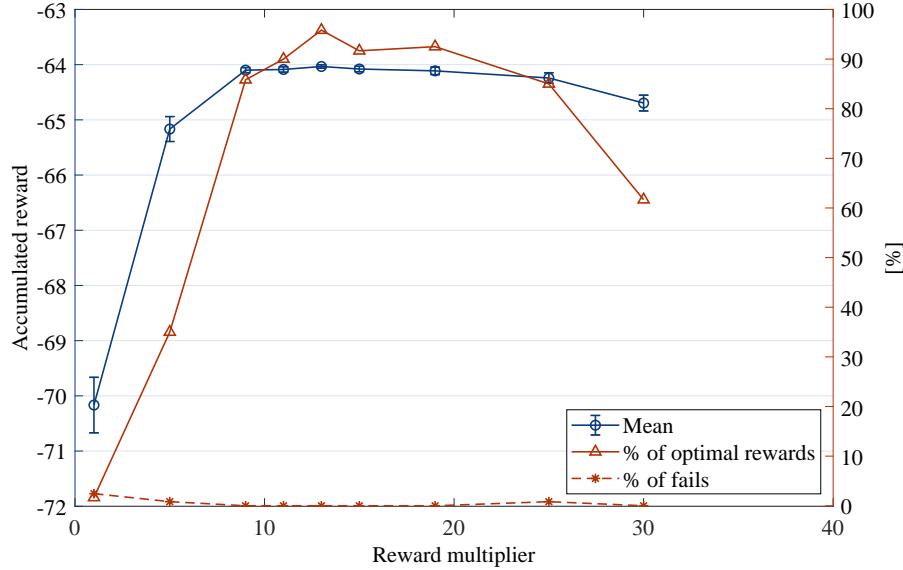
Figure 16: Reward multiplier's impact on the agent's performance.

the choice of 6500 maximum number of episodes, in the Fig. 17 are plotted the agent's performances for various values of maximum number of episodes.

As expected, a reduction in the maximum number of episodes leads to a lower percentage of optimal rewards. Therefore, in order to guarantee the threshold of 95% the value of the maximum number of episodes is maintained. After the previous scenarios experiments, it is possible to conclude that the best set's parameters are the ones expressed in the Table 12, with which the agent, in 120 experiments with 6500 episodes, was able to learn one of the 2 optimal assembly sequences 115 times ($\approx 95.8\%$) (Fig. 17), one of the assembly sequences with the second best accumulated reward 4 times ($\approx 3.3\%$) and one of the fifth best accumulated reward once ($\approx 0.8\%$), while never failing to learn a feasible assembly sequence.
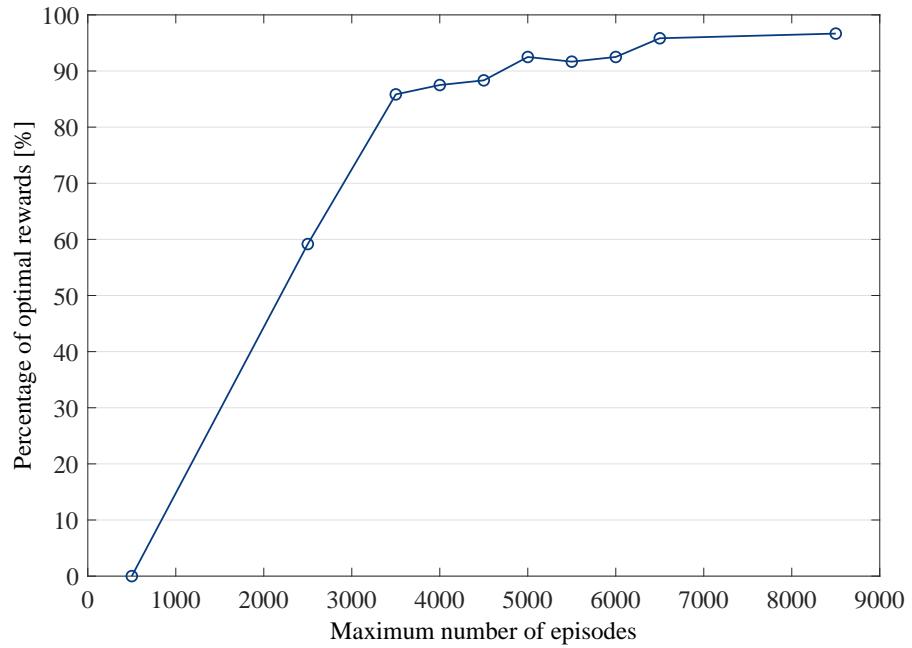
Figure 17: Evolution of the percentage of optimal rewards in regards to the maximum number of episodes.

Table 12: Optimal set's parameters.

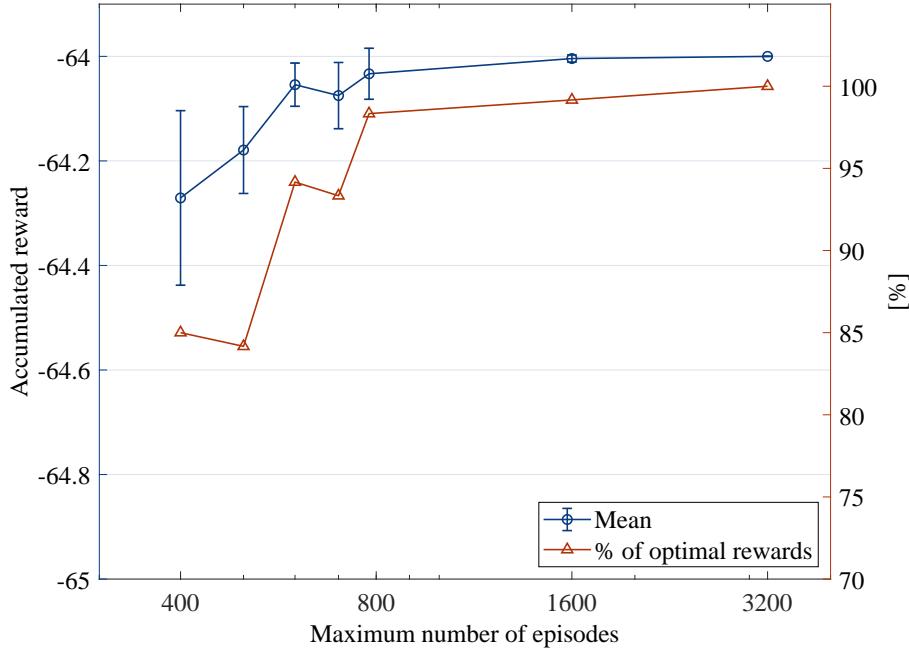| Parameter | Value |
| --- | --- |
| Learning rate | 1 |
| Discount factor | 1 |
| Epsilon | 0.9 |
| Epsilon decay | 0.00005 |
| Max steps per episodes | 15 |
| Max episodes | 6500 |
| Reward shift | 9 |
| Reward multiplier | 13 |
| Reward penalty | -1000000 |

Figure 18: Evolution of the mean accumulated reward and the percentage of optimal rewards in regards to the maximum number of episodes.

## 3.6 Q-learning algorithm - Scenario III: Learning an assembly sequence based on measured task average times and estimated variances with restricted actions

As previously discussed, the agent is capable of selecting impossible actions, which are penalised. However, this greatly increases the exploration required to achieve a correct assembly sequence. Therefore, in this final scenario, impossible actions were restricted, instead of being penalised in order to identify the possibility of reducing the number of episodes. Apart from the maximum number of episodes, which had the values of (400, 500, 600, 700, 750, 780, 1600, 3200), the epsilon decay, which had the corresponding values calculated from the regression equation in the Fig. 10 and the reward penalty that is now nonexistent the parameters used in the experiments are displayed in Table 12, Fig. 18.

It can be observed that by restricting impossible actions the maximum

number of episodes can be reduced to 780 episodes, which corresponds to only 21.7% of all possible assembly sequences, while maintaining a percentage of optimal rewards of 98.3%.

# 4 Conclusions

In this work, the challenges in the application of a reinforcement learning algorithm in a assembly sequence problem are studied, considering the implementation of a Q-Learning model-free algorithm. By formulating the problem as a MDP, it is shown that Q-Learning finds an optimal state-action policy that maximises the accumulated reward over a succession of given steps. This allows to verify the application of a scalable method to address the optimisation of an assembly sequence of an object as a sequential decision process, where the action in one state influences the transition to the subsequent state. Despite its recent application in the literature, RL methods show a straightforward versatility in complex problems where uncertainty plays a significant role, such as on-line industrial environments, where until now mostly traditional exact and non-exact optimization approaches are considered. The improvement efficiency of the assembly time of complex products is often impractical due to the complexity of assessing all tasks combinations. This approach has the advantage of achieving suitable optimisation results, where the optimal assembly sequence was learned 98.3% of the times. To guarantee this threshold (over 95%), the algorithm requires 780 assemblies (episodes) to correctly learn the best assembly sequence, which is corresponds to approximately 21.7% of the number of feasible assembly sequences. It is acknowledged that an increasingly complex assembly processes might reveal an unpractical use of Q-Learning algorithm due to the "curse of dimensionality". In future work, the acquisition of the tasks' durations on-line during real assemblies will be considered.

# 5 Conflict of interest

The authors declare that there is no conflict of interest.

# 6 Declaration of Competing Interest

The authors report no declarations of interest.

# 7 Funding

# References

[1] Miguel Vieira, Tânia Pinto-Varela, and Ana P. Barbosa-Póvoa. A model-based decision support framework for the optimisation of production planning in the biopharmaceutical industry. *Computers and Industrial Engineering*, 129:354 – 367, 2019.

[2] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction.* The MIT Press, second edition, 2018.

[3] David Silver, Aja Huang, Chris J. Maddison, Arthur Guez, Laurent Sifre, George van den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, Sander Dieleman, Dominik Grewe, John Nham, Nal Kalchbrenner, Ilya Sutskever, Timothy Lillicrap, Madeleine Leach, Koray Kavukcuoglu, Thore Graepel, and Demis Hassabis. Mastering the game of go with deep neural networks and tree search. *Nature*, 529(7587):484–489, Jan 2016.

[4] David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, Yutian Chen, Timothy Lillicrap, Fan Hui, Laurent Sifre, George van den Driessche, Thore Graepel, and Demis Hassabis. Mastering the game of go without human knowledge. *Nature*, 550(7676):354–359, Oct 2017.

[5] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin Riedmiller, An-

dreas K. Fidjeland, Georg Ostrovski, Stig Petersen, Charles Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharshan Kumaran, Daan Wierstra, Shane Legg, and Demis Hassabis. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, Feb 2015.

[6] S. Fernandes. Reinforcement learning para problemas de otimização. 2019.

[7] Yi-Chi Wang and John M. Usher. A reinforcement learning approach for developing routing policies in multi-agent production scheduling. *The International Journal of Advanced Manufacturing Technology*, 33(3):323–333, Jun 2007.

[8] Jamal Shahrabi, Mohammad Amin Adibi, and Masoud Mahootchi. A reinforcement learning approach to parameter estimation in dynamic job shop scheduling. *Computers and Industrial Engineering*, 110:75 – 82, 2017.

[9] Jinling Leng, Chun Jin, Alexander Vogl, and Huiyu Liu. Deep reinforcement learning for a color-batching resequencing problem. *Journal of Manufacturing Systems*, 56:175 – 187, 2020.

[10] J. Huang, Q. Chang, and N. Chakraborty. Machine preventive replacement policy for serial production lines based on reinforcement learning. In *2019 IEEE 15th International Conference on Automation Science and Engineering (CASE)*, pages 523–528, 2019.

[11] Panagiotis D. Paraschos, Georgios K. Koulinas, and Dimitrios E. Koulouriotis. Reinforcement learning for combined production-maintenance and quality control of a manufacturing system with deterioration failures. *Journal of Manufacturing Systems*, 56:470 – 483, 2020.

[12] X. Ou, Q. Chang, J. Arinez, and J. Zou. Gantry work cell scheduling through reinforcement learning with knowledge-guided reward setting. *IEEE Access*, 6:14699–14709, 2018.

[13] Xinyan Ou, Qing Chang, and Nilanjan Chakraborty. Simulation study on reward function of reinforcement learning in gantry work cell scheduling. *Journal of Manufacturing Systems*, 50:1 – 8, 2019.

[14] Keijiro Watanabe and Shuhei Inada. Search algorithm of the assembly sequence of products by using past learning results. *International Journal of Production Economics*, 226:107615, 2020.

[15] Yun Geon Kim, Seokgi Lee, Jiyeon Son, Heechul Bae, and Byung Do Chung. Multi-agent system and reinforcement learning approach for distributed intelligence in a flexible smart manufacturing system. *Journal of Manufacturing Systems*, 57:440 – 450, 2020.

[16] Yu-Ting Tsai, Chien-Hui Lee, Tao-Ying Liu, Tien-Jan Chang, Chun-Sheng Wang, S.J. Pawar, Pei-Hsing Huang, and Jin-H. Huang. Utilization of a reinforcement learning algorithm for the accurate alignment of a robotic arm in a complete soft fabric shoe tongues automation process. *Journal of Manufacturing Systems*, 56:501 – 513, 2020.

[17] Sharath Chandra Akkaladevi, Matthias Plasch, Sriniwas Maddukuri, Christian Eitzinger, Andreas Pichler, and Bernhard Rinner. Toward an interactive reinforcement based learning framework for human robot collaborative assembly processes. *Frontiers in Robotics and AI*, 5:126, 2018.

[18] Harley Oliff, Ying Liu, Maneesh Kumar, Michael Williams, and Michael Ryan. Reinforcement learning for facilitating human-robot-interaction in manufacturing. *Journal of Manufacturing Systems*, 56:326 – 340, 2020.

[19] Kaishu Xia, Christopher Sacco, Max Kirkpatrick, Clint Saidy, Lam Nguyen, Anil Kircaliali, and Ramy Harik. A digital twin to train deep reinforcement learning agent for smart manufacturing plants: Environment, interfaces and intelligence. *Journal of Manufacturing Systems*, 2020.

[20] Liang Hu, Zhenyu Liu, Weifei Hu, Yueyang Wang, Jianrong Tan, and Fei Wu. Petri-net-based dynamic scheduling of flexible manufacturing system via deep reinforcement learning with graph convolutional network. *Journal of Manufacturing Systems*, 55:1 – 14, 2020.

[21] Jens Kober, J. Andrew Bagnell, and Jan Peters. Reinforcement learning in robotics: A survey. *The International Journal of Robotics Research*, 32(11):1238–1274, 2013.

[22] Richard Ernest Bellman. *Dynamic Programming*. Dover Publications, Inc., USA, 2003.

[23] B. Calli, A. Singh, A. Walsman, S. Srinivasa, P. Abbeel, and A. M. Dollar. The ycb object and model set: Towards common benchmarks for manipulation research. In *2015 International Conference on Advanced Robotics (ICAR)*, pages 510–517, 2015.

[24] B. Calli, A. Walsman, A. Singh, S. Srinivasa, P. Abbeel, and A. M. Dollar. Benchmarking in manipulation research: Using the yale-cmu-berkeley object and model set. *IEEE Robotics Automation Magazine*, 22(3):36–52, 2015.