Dynamic Task Graphs for Teams in Collaborative Assembly Processes

Ana Macedo[a], Liliana Antão[b], João Reis[c] and Gil Gonçalves[d]
Faculty of Engineering, University of Porto, Rua Dr. Roberto Frias, 4200-465 Porto, Portugal

Keywords: Collaborative Robotics, Task Allocation and Scheduling, Task Graph, Reinforcement Learning.

Abstract: Collaborative robots are increasingly used in assembly processes, particularly in teams (Human-Robot or Robot-Robot), expanding the complexity and possible alternative sequences of operation and ways of team allocation to complete the assembling of a product. With this complexity, representing the possible sequences of actions needed to complete the task and the necessary constraints in a graph would improve the flexibility provided by team collaboration. However, the best sequence must be selected to increase productivity and profit, which is still challenging for a robot. This work proposes a modular system composed of three different components that, in a closed-loop interaction, allows a robotic agent to correctly plan a task given a set of operations, and optimize the task sequence allocation and scheduling plan. The effectiveness of the system is evaluated within an assembly process of different types of furniture for task sequence and allocation. The agent was able to converge successfully in three assembly scenarios: a table with 1 leg, a table with 2 legs and a table with 4 legs. Moreover, in the task allocation tests, the robotic agent was able to select actions according to the human operator expertise and its impact in the task completion time.

1 INTRODUCTION

With Industry 4.0, intelligent manufacturing and industrial robots emerged, which successfully perform repetitive tasks with high precision, boosting efficacy and productivity. However, it is challenging for a robot with limited cognitive abilities to complete complex tasks independently and meet high product variety and mass customization demands (Zhang et al., 2022). On the other hand, it is also impractical for humans to perform a task in a way that robots encode task models (Mangin et al., 2022). Thus, new needs emerge with the increasing necessity of collaboration between humans and robots. The production engineering society is giving considerable attention to collaborative systems, particularly in assembly processes, since they expand the possible alternative sequences of operations to perform and ways of team allocation to complete the assembling of a product, taking advantage of the fact that the tasks can be divided into more specific and discrete operations. Given this, creating a graph that represents the possible sequences of operations necessary to complete a task along with its constraints would provide more flexibility to team collaboration. In such a representation, any team member could easily observe the flow of actions it or its partner must execute, being able to formulate a complete sequence of steps to correctly perform a given task. However, the best sequence of operations planned for a specific product and partner must be selected, and robots do not have the human capacity to plan the best way of completing a task with a partner. The main motivation of this work is to contribute with modular and simple solution in the field of team collaboration in assembly processes. This approach focuses on adapting an agent's decisions automatically and intelligently, by choosing the sequence of operations to be performed within a given task.

Given the problem and the motivation, the main objectives of this work are to: (1) to create a modular, scalable system that includes the task representation in a **graph**, Machine Learning methods for **learning** and the task **simulation**, (2) build a graph that represents the sufficient and necessary information for the agent to efficiently learn, (3) use **Machine Learning** methods to teach an agent to correctly choose the task sequence in an assembly processes, (4) visualize and update the task status in the graph during the simulation, so that the operator can easily visualize

[a] https://orcid.org/0000-0003-4479-006X
[b] https://orcid.org/0000-0002-2903-0989
[c] https://orcid.org/0000-0002-1986-8366
[d] https://orcid.org/0000-0001-7757-7308

Macedo, A., Antão, L., Reis, J. and Gonçalves, G.
Dynamic Task Graphs for Teams in Collaborative Assembly Processes.
DOI: 10.5220/0011628500003393
In *Proceedings of the 15th International Conference on Agents and Artificial Intelligence (ICAART 2023) - Volume 1*, pages 17-25
ISBN: 978-989-758-623-1; ISSN: 2184-433X

the whole assembly process and be guided to perform corresponding tasks, and (5) compare the solution's performance between different algorithms, rewards and parameters.

We can divide this paper into four main parts. The first, including Sections 2 and 3, introduces the reader to the domain of this work through a review and analysis of related and available concepts. The second part, Sections 4 and 5, presents the modular framework and defines a case scenario. Section 6, the third part, develops the system and defines the procedures. Finally, Section 7 presents the experiments and the effectiveness of the proposed method is analysed, and Section 8 corresponds to the final conclusions.

2 BACKGROUND

This chapter presents an overview of the main concepts related to this work's scope, including Reinforcement Learning (RL) and Deep Reinforcement Learning (DRL), as well as their algorithms Q-learning and Deep Q-learning (DQL), respectively.

2.1 Reinforcement Learning

In RL there is a **closed-loop** interaction between the agent and the environment, where the actions taken by the agent based on the current state of the environment influence the next state. At each action made by the agent, the environment responds with a reward and the next state. A reward is based on a trial-and-error method, where a favorable output is 'reinforced', and a non-favorable output is 'punished'. The agent's final goal is to maximize the cumulative reward in the long run. RL problems are commonly formulated as a **Markov Decision Process** (BELLMAN, 1957) in the form of a tuple (S, A, $P_a(s_t, s_{t+1})$, $Ra(s_t, s_{t+1}), \gamma$) where: S is a finite set of states;A is a finite set of actions; $P_a(s_t, s_{t+1})$ is the probability to move to state s_{t+1} when choosing action a in state s_t; $R_a(s_t, s_{t+1})$ is the reward r_{t+1} for passing from state s_t to state s_{t+1} after taking action a, and γ is the discount factor.

A RL task is called a MDP if a state signal contains all the relevant information to make a decision, i.e. the next state s_{t+1} only depends on the immediately preceding state s_t and action a_t, and not on the previous ones, but at the same time is successfully informative. At each time step *t*, the agent observes the environment and reads the current state s_t in a finite set of possible states S. It selects an action a_t in a finite set of possible actions A at s_t, and takes it using actuators. At time *t + 1*, the environment returns the reward r_{t+1} and transitions to the next state s_{t+1}, upon which the agent relies for its future decisions.

RL must balance between **exploration** and **exploitation** to find the best sequence of actions. The agent must exploit the actions that maximize the reward, but also explore new actions that may lead to future better rewards; To find the balance, the right policy must be defined. **ε-greedy policy** is often used, where the action that maximizes the accumulated reward is chosen with an ε probability, otherwise a random action is selected.

One of the elements of RL problems is the **value function**. This element maps a state (or action-state pair) into the corresponding expected reward. Among all possible value functions for a state/state-action pair, there is one that leads to the maximum return. This function is called the **optimal value function**. The agent estimates the values for states or action-state pairs through experience. One of the most common methods for estimating these values is the Q-value, which relies on doing the average of n observed rewards for a given action a in a state s.

2.1.1 Q-Learning

In **Q-learning**, the goal is to learn the best next action given the current state. For that, the algorithm uses a **Q-table** to keep the values for each state-action pair. This table allows the agent to calculate the maximum expected future rewards for an action at each state.

Equation 1 defines the update function for the value of each state-action pair;

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

Where α is the learning rate ($0 < \alpha <= 1$), which defines how much the new value overrides the old value. If 0, the agent exclusively exploits prior knowledge, while if 1 the agent considers only explores, ignoring prior knowledge. The **discount factor** (γ) is a parameter used in the reward function that weights the immediate rewards over the future ones;

2.2 Deep Reinforcement Learning

In some decision problems, it is unfeasible to learn all possible sate-action pairs accurately with RL, so function approximators are used instead. **Deep Reinforcement Learning** (DRL) combines **Deep Learning** with **Reinforcement Learning**, by using deep neural networks (DNN) to learn useful **representation** in RL problems. This enables the algorithms to support **high-dimensional** and unstructured datasets,

2.2.1 Deep Q-Learning

To take advantage of learning through experience, but without worrying about space and time limits, Deep-

Mind (Mnih et al., 2015) researchers created a method that combines both RL and DL, called DQL. This technique estimates the Q-value of all possible actions for a given state using deep neural networks. The combination between RL and DL arise strong correlations between subsequent iterations. To overcome this issue, DQL provides a solution called Experience Replay, that stores the agent's past experiences into a dataset, where each experience is a four-tuple, (s_t, a_t, r_t, s_{t+1}), and extract a random mini-batch of these experiences to update the Q-values. To break the strong correlation, DeepMind (Mnih et al., 2015) researchers also propose a solution that is based on duplicating the Q-network, creating a copy called target network. The difference between the two copies are their parameters (weights) and how they are updated. While Q-network's parameters are trained, target network's parameters are periodically synchronized with the Q-network's parameters. The idea is that using the target network's values to train the Q-network will decorrelate the action-value with the target value and improve the stability of the network.

The agent selects an action according to an ε-greedy policy, which is represented via a DNN. Then, the same interaction between the agent and the environment, as in Q-learning method, occurs.

3 RELATED WORK

3.1 Collaborative Task Allocation Approaches

Research efforts on collaborative robots and tasks have been multiple in the past few years. *Zhang et al.* (Zhang et al., 2022) have designed a Human-Robot Collaboration (HRC) framework to assemble a product. The authors implement a dual-agent algorithm that extends the DDPG algorithm, where the actor-network of each agent processes the current state and outputs their own actions, and subsequently, the critic-network evaluates and calculates the value of the taken actions. The reward function is based on the difficulty and time each agent needs to complete the collaborative assembly task, with a structure tree to represent the assembly task.

Liu et al. (Liu et al., 2021) proposed a novel training approach adapted from the DQL method and called DADRL. This algorithms considers both the robot and the human as agents, where the robot is trained to learn how to make decisions, while the human agent represents the real human in HRC, demonstrating the dynamic and stochastic properties of the task. The aim is to teach the robot to be capable of decision-making and task-planning "under the uncertainties brought by the human partner using DRL", as the authors explain (Liu et al., 2021).

Yu et al. (Yu et al., 2021) formulate a human-robot collaborative assembly task as a chessboard game with specific assembly constraints determined by the game rules. The robot is trained with an algorithm based on DRL, where a CNN is used to predict the distribution of the priority of move selections and to know whether a working sequence is the one resulting in the maximum of the HRC efficiency.

Zhang et al. (Zhang et al., 2022), *Liu et al.* (Liu et al., 2021) and *Yu et al.* (Yu et al., 2021) have shown that DQL and DDPG-based algorithms are efficient in solving decision-making and sequence planning reducing problems and are efficient in complex tasks. Most of these works, however, do not approach graph structures and the advantages they might bring to collaboration in manufacturing sectors. *Liu et al.* (Liu et al., 2021) use a chessboard setting, but they assume this structure has limitations in representing some task constraints and relations.

3.2 Graph Structures for Task Allocation

In an assembly process, a task can be divided into smaller assembly subtasks, some of which are suitable to be performed by a robot and others are more complex and consequently require an operator to perform them (Zhang et al., 2022). Given this, building a graph representing the possible sequences of operations needed to complete a process, the operation's actor and other sufficient information about the process, is essential for an operator to perceive the task easily and for the robot to be efficient and effective in its assistance.

Mangin et al. (Mangin et al., 2022) developed a system that uses HTM to share information between the human and the robot and enable transparent communication and mutual understanding. This high-level hierarchical task representations is then transformed into low-level policies to teach the robot to successfully assist the human with supportive actions.

Murali et al. (Murali et al., 2020) proposed an architecture that aims to teach a robot to adapt reactively to unpredictable human operators's actions, while minimizing the overall process time. A collaborative task is represented as an "and/or" graph, where nodes correspond to states and hyper-arcs represent the possible actions the agent can perform to reach a particular state.

Yu et al. (Yu et al., 2020) map a human-robot collaborative assembly task into a chessboard game with

specific assembly constraints determined by the game rules, in which the chess piece move represents the decision made by an agent, considering the task structure and constraints. The authors train the robot with a combination of MCTS and a CNN. Using the chessboard structure, the completion time of each agent is formatted into a completion time matrix, which is given, along with all time matrices, as input to the CNN.

Hayes and Scassellati (Hayes and Scassellati, 2016) present a novel structure adapted from a traditional hierarchical structure, called CC-HTN, to represent the possible sequence of operations to complete a task. The goal is to give flexibility to the planning and allocation. From the CC-HTN, one can observe the ordered sequences (chains) and the unordered sequences (cliques). Nodes represent actions as subgoals and edges represent environmental constraints as compositions of subgoal completions.

All the solutions mentioned use hierarchical task structures, which have shown to be extremely powerful in supporting the planning of assembly tasks in HRC and co-robot collaboration. These studies, however, have some limitations: (Mangin et al., 2022) do not demonstrate results with complex tasks, not knowing if the MC solver is efficient in collaborative task planning; (Murali et al., 2020) and (Hayes and Scassellati, 2016) do not approach how the task sequences and the optimal one are obtained, with the first team considering them to be provided or computed *a priori*. The chessboard structure proposed by *Yu et al.* (Yu et al., 2021) presents some challenges in representing the task conflicts and other complex task relations.

4 PROPOSED SOLUTION

The goal of this project is to develop a system capable of automatically generating efficient assembly sequences that respect precedence dependencies and task pre-conditions in a collaborative environment, while transmitting knowledge on the task in a graph structure.

The proposed solution of this work is then an architecture divided into three modules, which, directly or indirectly, influence the performance of each other. In the first module, the **graph** aggregates information about the task (**subtasks** and their **constraints**) that will be perceived by the agent to define the observation and action spaces on which the agent will run the algorithm, as well as the **possible sequences** that will be taken into account by the reward function of the agent, so it penalizes actions that do not meet precedence dependencies. The graph also reports the **average time** each **operation** takes to be completed by each actor, which will influence the learning process for task allocation. The **agent**, in the second module, tries to successfully select the best sequence of actions, using **off-policy RL** algorithms. The training process ends with a Q-table that returns the action that generates the maximum accumulated reward for each state. Regarding the third module, the robotic manipulator, at each step, will **simulate** the **optimal action** and update the status of the task in the graph, particularly the actions that are completed, that are being performed and that are still to be done. The simulation also allows an agent to perceive the execution time of each action, as well as its actor. The closed-loop interaction between the three modules is represented in Fig. 1.

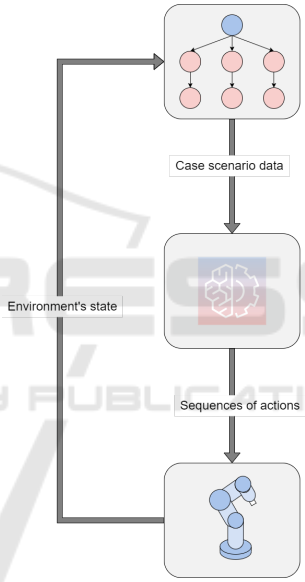


Figure 1: Flow of the architecture proposed as a solution.

The overall sequence result is validated using the robotic simulator Coppelia Simulator, formerly known as V-REP (Rohmer et al., 2013).

5 CASE STUDY

A furniture assembly task was chosen as a case scenario, as it depicts a complex process that can result in different final products. In particular, we focus in the task of **assembling tables** as a case scenario. The structure of the table was based on a design presented by *Zeylikman et al.* (Zeylikman et al., 2017), which constituent parts can be used to create a variety of different objects, such as chairs, shelves and consoles,

providing modularity and scalability to our model.

The table is comprised in 5 structural parts, **one base** and **four legs**, and composed by 15 components. A table leg consists of a dowel and one F-type bracket, and the base needs one plywood. The T-type brackets are used to assemble the base with the legs, and the screws are the fasteners, which are fastened by a screwdriver. Given its constituent parts, the task can be easily **discretized** into 27 smaller and simpler problems, called **subtasks**. This division is the **key** a good representation of the task on a graph and consequently a better understanding of the different possible sequences and the assigned resources.

We created two simpler scenarios called **two-leg-table assembly task** and **one-leg-table assembly task**, where a table only has two legs and is decomposed in **15 subtasks** in the first scenario and has one leg and is decomposed in **9 subtasks** in the second one. The idea is to create tasks that are less time-consuming for the algorithm.

Each subtask of our tasks must be further decomposed into a sequence of **operations**, containing more relevant and specific information about the task, such as picking, placing, snapping, positioning, and fastening an object. The first four operations are based in the same basic and common movements a robot may perform in manufacturing: move the gripper to a position, grasp an object and/or release an object. Some operations, such as fastening, require high precision and are too complex for a robot to perform.

In the manufacturing industry, operators all have different experience and knowledge about a task, which influences the time each one takes to perform an operation, as well as the probability of failure. It is therefore essential that operators can be differentiated by their level of expertise: **beginner**, **intermediate** or **expert**. An expert, as opposed to a beginner, in average, performs the tasks with a lower time and a lower variance.

6 IMPLEMENTATION

This section aggregates the implementation procedures of each module to build our solution. The entire system was developed in Python, using several packages like *Networkx* to to construct and draw graphs.

6.1 Task Graph Module

We define our tasks' graph structure as a simple and appealing **hierarchical task structure**, close to human intuition that has shown good performance in allowing the agents to communicate and understand each other's actions. Our graphs are composed of nodes and links, where the root, representing the task, is divided into subtasks, which in turn are divided into the sequence of concrete actions. Each node of our graph is composed by the following three elements:

- **id:** unique node identification;
- **type:** classifies the node as *task*, *subtask* or *operation*;
- **state:** determines whether the node is still to be, is being, or has already been executed. It allows the agents to perceive the task's evolution;

Operation type nodes, in addition to these, contain four more parameters:

- **operation:** indicates which action the node refers to (*pick*, *place*, *snap*, *position* or *fasten*). Its value must then be a string;
- **object:** indicates which object the node refers to, e.g. screw (string value);
- **time_robot:** Estimated time of operation's execution by the robot (integer value);
- **time_operator:** Estimated time of operation's execution by the robot, depending on his/her expertise (integer value).

The sub-task and task type nodes may also include an additional element, **sequence**, that expresses whether the node is *sequential*, *parallel* or *alternative*, i.e. whether a task's subsequent sub-tasks or a sub-task's subsequent operations might be done at the same time by two different agents (**parallel**), must be executed by a specific order of execution (**sequential**) or must be performed exclusively (**alternative**). This set of elements demonstrates the type of constraints that may exist when executing a collaborative task along with the complexity of the collaboration in a human-robot or a robot-robot team. In our graphs, the parallel operation is represented by ——, the sequential by $\rightarrow$ and the alternative by **V**.

The overall graph structure for the table assembly task is illustrated in Fig. 2, simplified for better and understandable visualization.

6.2 Robot Intelligence Module

The observation space of our algorithm's MDP environment corresponds to the evolution of the task's status, i.e. each space is a tuple composed of **the number of completed subtasks** and **a list with the objects/parts in the workspace**. The last element is important, as objects and parts that are in the workspace influence the feasible sequences of actions. The initial state corresponds to no actions executed and no objects/parts in the workspace and the

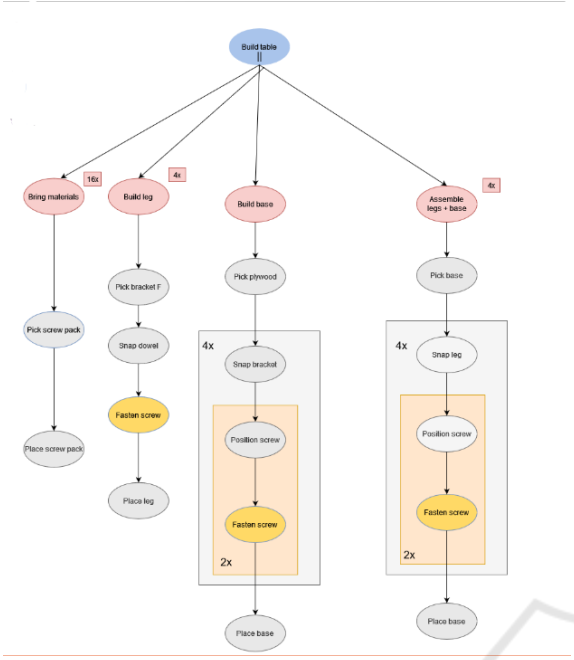


Figure 2: Graph structure for the table assembly task.

final state to all actions executed and all objects/parts in the workspace. An action in our MDP environment corresponds to a tuple composed of a sub-task and the actor who will perform the sub-task. We started by implementing **Q-learning** algorithm, since the Q-table provides us easier analysis and interpretation of the environment's behaviour and the overall training process results and evolution.

The system must generate only realistic scenarios, i.e. assembly sequences with no repeated actions and that satisfy precedence dependencies. Therefore, and since the agent's final goal is to maximize the cumulative reward in the long run, the system must penalize the sequences that do not meet the required conditions. On the other hand, to ensure that the task progresses, the environment must reward the status of the global assembly task to guide the agent through the decision-making. The states that do not satisfy precedence dependencies are still considered during the learning process, but penalized. The aim is to assign low rewards to those states, making the robot more robust to avoid undesired sequences. The final reward function is given in Equation 2:

$$r_t = \ln(S_E + bias) - PP - RP, \quad (2)$$

Where RP represents the penalty received for a selected repeated action, PP corresponds to the penalty given for not satisfying precedence dependencies, and ε is the bias of S_E, with a value of 0.1, to avoid the problem caused by null progress at the beginning of the task.

We started by focusing on a RL approach based on successive interactions with the environment to learn the Q-values and consequently find an optimal policy that maximizes the cumulative reward, more specifically, the **Q-learning** algorithm. This algorithm's Q-values matrix (Q-table) provides us easier analysis and interpretation of the training process results and evolution, as well as how the environment must behave for the algorithm to converge. After, we proceeded to implement DQL for the agent to complete an assembly task with high-dimensional space requiring less time and memory. The MDP environment on this DRL algorithm was defined using the OpenAI's **Gym** (Brockman et al., 2016) framework. Our observation and action spaces were represented as an **OpenAI Gym Discrete Space** (Openai, 2022) with n elements. In the observation space n corresponds to the total number of observation states, and in the action space n corresponds to the total number of subtasks.

6.3 Simulation Module

The agent simulates the sequence of actions for the given collaborative assembly task. But for that, it needs the **poses of each object**, which are given in a text file. Each operation the robot must perform is divided into one or several **specific and discrete movements** and sent to a previously implemented API connected, which decodes the movements and sends them, through sockets, to the V-REP simulator.

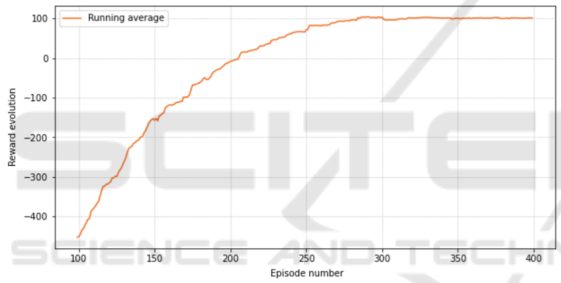
The simulation module introduces a new node type to the graph, the **wait node**, which represents the time "lost" between an exchange of actors in the execution of a subtask. During the simulation, the graph nodes are colored with different colors depending on the node status, i.e. the respective task/subtask/operation has not been performed, is in progress, or is finished.

In addition to a collaborative assembly, our tasks can also be performed manually or automatically, i.e. the task is completely performed by the human agent or executed only by the robot (with the exception of the fastening operation, which only can be done by the human operator). In these two options, when an agent fails the performance of an operation or there is an external error, the graph suggests the agent to repeat the action. In the collaborative option, failures are not addressed.

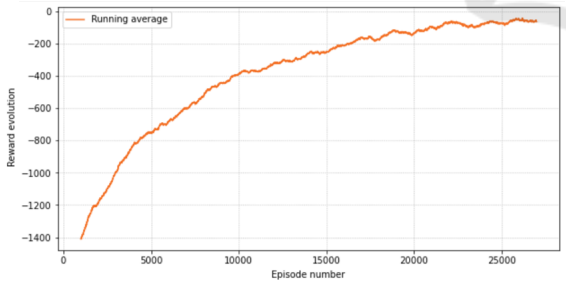
In the end of our simulation, we have a system composed of three modules, where each module feeds the others, directly or indirectly, with the necessary data for they to successfully run.

7 EXPERIMENTS AND ANALYSIS

We started by training the agent to learn **feasible assembly sequences**, with no prior knowledge, with **Q-learning** for the tasks with 15 and 9 actions. We run 100 trials for each experiment and compare the training efficiency through the running average reward evolution and the percentage of **possible assembly sequences** learned. After trying several sets of reward weights and hyperparameter values, the agent was able to converge the reward in 400 episodes with a weight of 20 for each reward parameter, α and γ defined with values of 0.99 and the exploration rate decaying by 0.01, for the **one-leg-table task**. And the minimum number of episodes from which we can train the agent to complete two-leg-table task was **27 000 episodes**, with an exploration rate decay of **0.001** and the same weights for the reward parameters. Fig. 3 illustrates the reward evolution for the two processes where the agent was successful in completing the with feasible sequences in 100% of the experiments.



(a)



(b)

Figure 3: Running average reward evolution for (3a) the one-leg-table assembly task with 400 episodes and an exploration rate decay of 0.01, and (3b) the two-leg-table assembly task with 27 000 episodes and a exploration rate decay of 0.001.

The **balance between exploration and exploitation**, represented by the exploration rate which was decayed at each episode by the epsilon decay, has shown to have a large impact on the agent's success. A lack of exploration led to a early reward stagnation. On the other hand, excessive exploration requires more episodes for the agent to learn and the training gets more time consuming. The **reward weights** has also proven to significantly influence the learning results. The penalties' weights must be at least equal to the weight of the progress state for the agent to learn which state-actions pairs lead to higher rewards. The processes trained with higher reward penalties return lower rewards in the first episodes, but the overall convergence curve is sharper and the evolution is higher.

After learning possible assembly sequences, the goal is now to minimize the global time to complete the task. For that, we add a new penalty to the reward function representing the completion time of each action, *SubtaskTime*. To evaluate the algorithm's decisions, three performance **metrics** are used:

- The **time** required to finish the global assembly task;
- The **operator's working time** on the assembly task;
- The **number of operations** assigned to the **operator**.

We ran 300 experiments, in which for every 100 executions the operator has a different **expertise** (beginner, intermediate and expert). We compared the percentage of actions assigned to each actor and the results are shown in Table 1.

Table 1: Percentage of actions allocated for each actor at each operator's expertise.

Operator's Expertise	Actor	Percentage of actions allocated
Beginner	Human	≅ 7.13%
	Robot	≅92.87%
Intermediate	Human	≅ 50.9%
	Robot	≅49.1%
Expert	Human	≅96.0%
	Robot	≅4%

As expected, even considering the time taken to switch between actors when the operator is required to fasten screws, the robot was assigned to more actions if its human partner is a beginner, since the robot takes, on average, less time to complete the task. An intermediate operator has a robot-like assembly speed and hence both agents have a similar percentage of subtasks allocated. An expert, on the other hand, has more experience and ease in performing assembly tasks and is assigned to more actions.

Table 2 details the **mean** and **standard deviation** of the **global task** completion time with or without

the robot learning task allocation, and also considering each operator's expertise. We can see that without learning task allocation, the global assembly duration is higher and has more variance than with the agent learning task allocation. This happens because without considering the time, the agent assigns the actor randomly, instead of assigning the one that executes the actions in less time. As the expertise of the operator increases, the mean duration of the task, considering optimal task allocation, decreases, as expected. The standard deviation of task time in the experiments with a beginner operator and with learning task allocation is lower, because the operations are mostly assigned to the robot, which is more constant, due to its automation, than a human in performing tasks.

Table 2: Mean and standard deviation of the global task duration, in seconds.

Operator's Expertise	Task allocation	Mean	Standard deviation
Beginner	Yes	135.2	0.23
	No	143.22	3.46
Intermediate	Yes	126.58	0.42
	No	128.04	1.26
Expert	Yes	73.25	0.42
	No	80.7	5.32

For the **table assembly task**, with 27 actions, we obtained the results from Fig. 4, training the agent with the **DQL** algorithm. The reward seems to be converging until around the episode number **20 000**, although with more episodes it reduces the reward value.

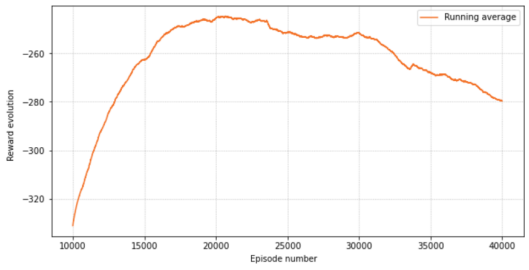


Figure 4: Running average reward evolution for the table assembly task with Deep Q-network.

The expected results, given the high-space dimensionality of the case study (1 812 473 states) would be a faster convergence with the DQL algorithm. However, this was not verified. From our intuition, this happened due to the complexity and number of parameters that are needed to optimize, in the DQL network, which due to time constraints it was not possible to evaluate the impact of various combinations of parameters, thus not being possible to achieve the optimal network that would lead to a faster convergence by this algorithm.

8 CONCLUSION

A novel system has been proposed to tackle the problem of sequence allocation and scheduling optimization in collaborative processes. This system trains an agent with RL and DRL approaches and simulates the process on top of a task graph represented as a combination of hierarchical structures and directed graphs. Two types of experiments were performed: the first with the aim of generating feasible sequence of actions to complete the task given precedence dependencies, and the second with goal of task allocation to an actor. The system was able to converge successfully in 3 scenarios: a table with 1 leg, a table with 2 legs and a table with 4 legs, in 400, 27 000 and 20 000 episodes, respectively. We could obtain a system that acquires data from the graph and sends back decisions to trigger nodes. The graph allow us to visualize the sequences triggered by the robot, from the beginning of the process to its conclusion. We obtained an algorithm that makes decisions about sequences of operations and allows the team to complete the task collaboratively and obtain the final product. Last but not least, We have a simulation through which we can validate if the process is correct.

In the future, the model developed in this paper can be extended to multi-agent planning, which will allow simultaneous execution of sequences between the two agents, and consequently improve the solution's efficiency and application in real-life scenarios. Additionally, we will further address failures within collaboration, making the robot capable of compensating for an error in a sequence. Our solution would have more valuable results if it was tested with a real UR5 collaborative robot arm, increasing the transferability to real-world applications.

ACKNOWLEDGEMENTS

INDTECH 4.0 – New technologies for intelligent manufacturing. Support on behalf of IS for Technological Research and Development (SI a Investigação e Desenvolvimento Tecnologico). POCI-01-0247-FEDER-026653

REFERENCES

BELLMAN, R. (1957). A markovian decision process. *Journal of Mathematics and Mechanics*, 6(5):679–684.

Brockman, G., Cheung, V., Pettersson, L., Schneider, J., Schulman, J., Tang, J., and Zaremba, W. (2016). Openai gym. *CoRR*, abs/1606.01540.

Hayes, B. and Scassellati, B. (2016). Autonomously constructing hierarchical task networks for planning and human-robot collaboration. In *2016 IEEE International Conference on Robotics and Automation (ICRA)*, pages 5469–5476.

Liu, Z., Liu, Q., Wang, L., Xu, W., and Zhou, Z. (2021). Task-level decision-making for dynamic and stochastic human-robot collaboration based on dual agents deep reinforcement learning - the international journal of advanced manufacturing technology.

Mangin, O., Roncone, A., and Scassellati, B. (2022). How to be helpful? supportive behaviors and personalization for human-robot collaboration. *Frontiers in Robotics and AI*, 8.

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M., Fidjeland, A. K., Ostrovski, G., and et al. (2015). Human-level control through deep reinforcement learning.

Murali, P. K., Darvish, K., and Mastrogiovanni, F. (2020). Deployment and evaluation of a flexible human-robot collaboration model based on AND/OR graphs in a manufacturing environment. *CoRR*, abs/2007.06720.

Openai (2022). Gym/discrete.py at master · openai/gym.

Rohmer, E., Singh, S. P. N., and Freese, M. (2013). Coppeliasim (formerly v-rep): a versatile and scalable robot simulation framework. In *Proc. of The International Conference on Intelligent Robots and Systems (IROS)*.

Yu, T., Huang, J., and Chang, Q. (2020). Mastering the working sequence in human-robot collaborative assembly based on reinforcement learning. *IEEE Access*, 8:163868–163877.

Yu, T., Huang, J., and Chang, Q. (2021). Optimizing task scheduling in human-robot collaboration with deep multi-agent reinforcement learning. *Journal of Manufacturing Systems*, 60:487–499.

Zeylikman, S., Widder, S., Roncone, A., Mangin, O., and Scassellati, B. (2017). The HRC model set for human-robot collaboration research. *CoRR*, abs/1710.11211.

Zhang, R., Lv, Q., Li, J., Bao, J., Liu, T., and Liu, S. (2022). A reinforcement learning method for human-robot collaboration in assembly tasks. *Robotics and Computer-Integrated Manufacturing*, 73:102227.