DQN as an alternative to Market-based approaches for Multi-Robot processing Task Allocation (MRpTA)

Paul Gautier, Johann Laurent, Jean-Philippe Diguet

▶ **To cite this version:**

Paul Gautier, Johann Laurent, Jean-Philippe Diguet. DQN as an alternative to Market-based approaches for Multi-Robot processing Task Allocation (MRpTA). International Journal of Robotic Computing, inPress, 3 (1). hal-03251554

HAL Id: hal-03251554
https://hal.science/hal-03251554

Submitted on 7 Jun 2021

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Preprint for IJRC

DQN as an alternative to Market-based approaches for Multi-Robot processing Task Allocation (MRpTA)

Paul Gautier[1], Johann Laurent[1], and Jean-Philippe Diguet[2]

[1] Lab-STICC UMR CNRS 6285, Université de Bretagne Sud, Lorient, France
{firstname.author,second.author}@univ-ubs.fr
[2] CROSSING IRL CNRS 2010 CROSSING, Adelaide, Australia.
Jean-Philippe.Diguet@cnrs.fr

Abstract. Multi-robot task allocation (MRTA) problems require that robots make complex choices based on their understanding of a dynamic and uncertain environment. As a distributed computing system, the Multi-Robot System (MRS) must handle and distribute processing tasks (MRpTA). Each robot must contribute to the overall efficiency of the system based solely on a limited knowledge of its environment. Market-based methods are a natural candidate to deal processing tasks over a MRS but recent and numerous developments in reinforcement learning and especially Deep Q-Networks (DQN) provide new opportunities to solve the problem. In this paper we propose a new DQN-based method so that robots can learn directly from experience, and compare it with Market-based approaches as well with centralized and purely local solutions. Our study shows the relevancy of learning-based methods and also highlight research challenges to solve the processing load-balancing problem in MRS.

Keywords: MRTA; distributed system; reinforcement learning; deep Q-learning

1 Introduction

The robust and flexible nature of multi-robot systems (MRS) makes them particularly suitable for critical tasks such as security or rescue missions. However, effectively coordinating the robots requires to accurately distribute the work within the system which leads to the multi-robot task allocation (MRTA) problem. This problem has been extensively studied and many approaches have been proposed to solve it [2].

But autonomy, safety and accuracy requirements come with a high computing cost due to a significant increase of sensor infrastructure and of algorithm complexity developed to sense, analyse and decide. A direct consequence is that

processing tasks, within the MRS, become a major impediment to their development. In fact, despite the improvement of embedded-systems processing and storage capacities, computing resources are intrinsically limited. Cloud-robotics [7], [15] have been introduced as an alternative solution, but it is applicable only when reliable and high bandwidth connections are available and when the processing latency, namely the response time, is not critical.

The concept of Robotic Cluster has also been introduced to speed up computationally hard tasks such as SLAM, which is detailed in [10] and [3]. This work demonstrates the possibility to get benefit from multiple processing resources distributed over a cluster of interconnected robots to execute a parallel version (e.g. multithreaded) of a complex application task. If we extend this work to multiple independent tasks, then sharing resources allows to improve the processing capacity of each single robot with multi-tasking capabilities. Considering a set of processing tasks to be executed by the MRS, the new question to solve becomes a task allocation problem with constraints such as execution time and priorities. This question is a variant of the MRTA problem focused on processing tasks (MRpTA).

Considering the MRS context, different options are possible. Firstly, the method can be centralized or decentralized. On the one hand, a centralized approach solves most of the difficulties by allowing a decision making based on a perfect knowledge of the state of each processing resource. But on the other hand, it brings two major drawbacks. The first one is the unreliability since the central agent introduces a single point of failure. The second one is the inherent communication overhead due to the aggregation of the all system knowledge in a single robot. This communication cost strongly limits the system scalability and makes it unpractical for MRS. Therefore, we only focus on distributed systems. Regardless the architecture the system uses, the problem may take many forms from travelling salesman problem (TSP) to job scheduling. The latter is more frequently studied in the field of multiprocessors or computer clusters than in robotics. It also raises new challenges related to the rapid development of autonomous systems. Indeed, the last few years have seen the emergence of new heavy processing methods in several areas such as computer vision, sensor fusion and especially machine (deep) learning.

In this paper, we investigate the task allocation problem within a MRS according to their computational and memory costs, with the joint objectives of task completion and fair load balancing.

A market-based approach is a quite straightforward possible candidate, which has been used in conventional MRTA problem in robotics and that can be adapted to the specificity of processing tasks as detailed in Sec. 2. But a MRS is also composed of increasingly complex embedded systems, including multiple sensors, multi-core architecture with GPU, which are almost impossible to accurately model. Moreover the robots evolve within an uncertain environment and execute applications accordingly. The dynamic nature of the whole system requires a high degree of adaptability to cope with the lack of information (data availability, communication errors) and rapid changes (failure, object detection).

Based on these observations we consider the opportunity to use machine learning approaches and more specifically Reinforcement Learning (RL), which has been successfully applied in many fields, such as energy management, communication optimisation, job scheduling, etc. In RL, the agent progressively learns to improve the quality of its decisions according to the experience it acquires by means of a reward that reflects the efficiency of the chosen actions. Recently, the combination of deep neural networks and RL has been introduced (Deep Q-Learning) to deal with the scalability issue of RL. Thus, instead of storing values in tables that grow with the environment dimensions, a neural network is used to approximate the policy function and find the best action according to the environment. This method has become very popular with the success of the Deep Q-Learning on Atari games [12].

In this article, we explore the viability of Deep Q-Learning to solve the MRpTA problem and compare it with a multi-robot task allocation problem with the following questions:

- Can robots of a fully decentralized system learn to efficiently manage task allocation on their own?
- What is the performance of a method based on Deep Q-Learning compared to a more traditional approach like the market-based one?
- What does the use of learning imply for real-life applications?

The rest of this paper is organized as follows: Sec. 2 discusses relevant works on the MRTA problem and RL. Sec. 3 explains the modelling of our problem and Sec. 4 the approach used to solve it. Sec. 5 describes our experimentation set-up. Experimental results are discussed in Sec. 6. Finally, we conclude and introduce future work based on this study.

2 Related work

2.1 Multi-robot tasks allocation

We briefly introduce the key points of our MRTA problem, the reader can refer to cited references for detailed surveys and complete formulations.

MRTA architecture The literature provides various instances of the MRTA problem, in order to offer a broader and more theoretical view, Gerkey and Mataric [2] have proposed a taxonomy for those problems. Based on their definition, our architecture is a multi-task type, single robot and instantaneous assignment (MT-SR-IA). In this decentralized context, we compare the efficiency of two methods: a market-based approach and an approach using reinforcement learning.

Market-based systems A market-based approach enhances the efficiency of the overall system by maximizing individual profits. Several surveys address this subject [2] and [6]. In our comparison cases, overloaded robots sell tasks to others

in order to maximize the system's task completion rate. The sale is made by auction where an auctioneer (the seller) offers a task for sale. Each participating robot submits an auction whose valuation depends on its ability to perform the task. The best bidder wins the task and runs it. The model used for auctions is sequential single item mechanism, which is both efficient and inexpensive [9]. Further details on the auctions can be found in [14] and [16].

Although it is inexpensive in processing, this process raises the question of bid estimation. It is indeed difficult to correctly estimate a bid when several parameters are involved and especially, when their value depends on the environment's state. Moreover Market-based approaches require multiple exchanges including a unique auctioneer that introduce an additional delay. To overcome these issues, we propose to explore another approach based on reinforcement learning.

2.2 Reinforcement learning

One of the main goals of our RL approach is to remove the auction system. Agents must then be able to correctly estimate the relevance of a transfer without exchanging information about their status.

Principles

In RL, the agent observes at each time step, the environment's state s_t and chooses an action a_t. This action modifies the environment, which then proceeds to the next state s_{t+1}. Then, the agent receives a reward r_t according to the quality of its choice. The learning aim of the agent is to maximize the cumulative value of future rewards. To operate, this method requires that the state transitions are stochastic and have the properties of a Markov Decision Process (MDP). It means that the rewards r_t and states' transitions s_{t+1} must depend only on the environment s_t and the action a_t. Fig 1 illustrates the principle of reinforcement learning

One of the most popular reinforcement learning methods is Q-Learning, which chooses its actions based on Q-values. Q-Learning uses a table to store all Q-values of all possible {state, action} pairs. This Q-table is updated using the Bellman equation (eq. 1). The action selection is usually done with an ϵ-greedy policy. The Q value can be calculated using the following formula and definitions:

$$Q(s,a) = Q(s,a) + \alpha[y - Q_(s,a)] \tag{1}$$

where y denotes the temporal difference target:

$$y = r(s,a) + \gamma \max_{a'} Q(s',a') \tag{2}$$

- $r(s,a)$ returns the reward of action a in the state s
- $\gamma \in [0,1]$, the discount factor witch control the value of future rewards.
- α, the learning rate.
- $\max_{a'} Q_\pi(s',a')$ returns the optimal possible Q-value of the next state.

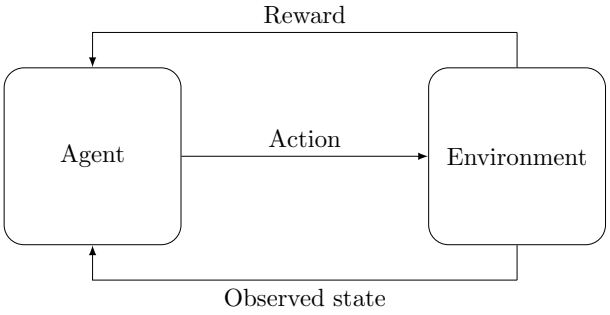

Fig. 1. Reinforcement Learning Diagram

Although effective, the Q-Learning method has some limitations. Indeed, it does not scale with the number of states since the number of pairs {state,action} increases exponentially leading to a very large Q-table and so requiring a large amount of memory. To overcome this problem, the Deep Q Network (DQN) method has been introduced [12].

Deep Q-Network

A DQN uses a deep neural network (DNN) as a function approximator to predict Q-values. An approximation is possible since an agent must take similar actions for 'close-by' states. Fig.2 shows a Deep Q Network who uses parameter fitting to construct a function able to predict Q-values.

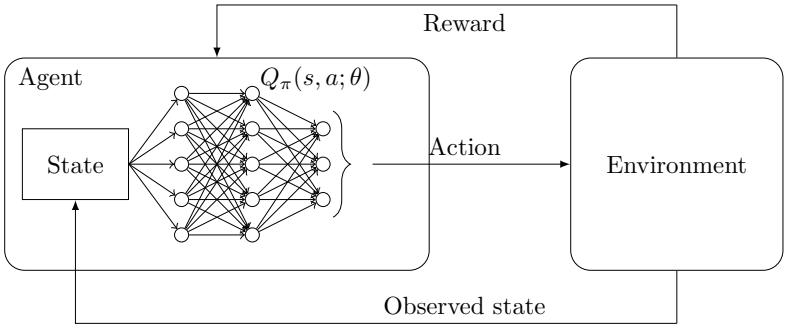

Fig. 2. DQN diagram

First, we construct a loss function using the mean square deviation to define the target function (eq. 3). Then we update weights using the Adam optimiser

[8], which is computationally efficient.

$$L_i(\theta_i) = E[(y_i - Q_\pi(s, a, ; \theta_i))^2] \tag{3}$$

with:

$$y_i = r + \gamma \max_{a'} Q(s', a'; \theta_{i-1}) \tag{4}$$

DQN improvements

However, replacing the function of estimating values Q by a DNN, leads to the instability of the algorithm. Indeed, the target is calculated using a network updated every each iteration leading the target to be non-stationary [13]. This problem can be overcome by using a second network of parameters θ^-, called the target network, to calculate the target (eq. 5). This network is a frozen copy of the first network updated every c the iterations by copying θ parameters.

$$y_i = r + \gamma \max_{a'} Q(s', a'; \theta^-) \tag{5}$$

Another problem arise from strong correlation between continuous states and action inputs. Each small update of a Q-value of an action causes the modification of the set of network weights, which affects the Q-value of each action in the other states. This strongly impacts the distribution of sampling data. To avoid instability we use *experience replay* developed by [13]. With this mechanism, each experiment {state, action, reward} is stored in a memory. The agent learns from samples selected randomly and consistently in its memory. Thus, all correlations are broken, and learning is accelerated since an experience can be used multiple times.

Both Q-learning and DQN suffer from a problem of overestimation of action values resulting from using the same network/parameters for the selection and evaluation of an action [5]. Double DQN (DDQN) solves this problem by using another network (the target network is a natural candidate) when evaluating [4] leading to :

$$y_i = r + \gamma Q(s', \arg\max_{a'} Q(s', a'; \theta_i); \theta^-) \tag{6}$$

Distributed reinforcement learning

Our approach being decentralized, each robot has its own learning system. Many distributed approaches take advantage of the multiplicity of local learning to accelerate the learning of the system [17]. Indeed, it is possible to exchange experiences between agents. Thus, each agent learns from others which allows it to converge more quickly towards an optimal solution as shown by [11]. We do not consider this mechanism that requires a large data exchange which is incompatible with a real deployment of mobile robots.

Hereafter we describe the method used to compare the Market and DQN approaches.

3 Problem Definition

The compared approaches share the same definition of the environment. They differ by their decision making process, in particular for the load balancing mechanism.

3.1 MRS definition

The MRS pursues a dual objective: the completion of a maximum of the tasks assigned to it while promoting high priority tasks. To do this, the system must distribute the load using the transfer mechanism. However, the constraints imposed by the environment make it impossible to meet both objectives. The system has to make a choice.

These choices are made in a decentralized context, thus each robot acts with a limited understanding of the situation based solely on local knowledge. Robots only communicate during a transfer, no further information exchange takes place. In order to get closer to real conditions, an area system is defined. The latter limits transfers to robots in the same area. During the course of the mission, the robots move between areas, but not of their own will. Changing area is not part of the actions they can take. These only affect tasks and are the following:

1. Run: the robot performs the task. For this, it must possesses enough available computing resources.
2. Postpone: the robot postpones the task. If the task has not yet reached its laxity value, it will be present in the robot's task queue at the next iteration. Otherwise, it is considered failed (and will disappear from the task queue).
3. Transfer: The robot tries to transfer the task to another robot in its area. In case of failure, the task is automatically postponed. Transferring a task involves generating an overhead.

The prerequisites of each action depend on the task characteristics on which the robot acts.

3.2 Task definition

In our model, tasks are independent (tasks with dependencies are grouped into a single task) and have a fixed priority (but no task is imperative). Furthermore, there is no task preemption except case given 4.1. Similarly to prior work [3], we assume that task characteristics is known upon its arrival in the robot queue. The main task features are the following:

Since pending tasks are stored in a queue and decision are made sequentially, agents cannot choose which task to deal with. To solve this problem, the tasks in the queue are automatically sorted by priority (1 to 3) and by earliest deadline.

Table 1. Main characteristics of a task

Characteristics	Definition	Value
CPU	Amount of CPU resources required to perform the task	Integer $0 \rightarrow 100$
Memory	Amount of memory required to perform the task	Integer $0 \rightarrow 100$
Execution time	Number of iterations required to complete the task	Integer $1 \rightarrow n$
Priority	Upon creation, each task receives selected fixed priority value	1 : High 2 : Medium 3 : Low
Laxity	Maximum possible delay (in iterations) before the task starts	Integer $1 \rightarrow l$

3.3 Time definition

Time modelling is key parameter for a simulator. In our study, the simulation of the system behavior requires to discretize the temporality into iterations of equivalent durations as illustrated in Fig. 3. However, the discretization granularity should not be too fine to minimize the number of unnecessary time steps (without changes) which considerably slow down the simulation. We manage this tradeoff with the following assumptions:

1. Robots receive simultaneously their new tasks and process them in parallel.
2. Several tasks start or end at exactly the same time.
3. The transfers resolution occurs after the local allocations, but in the same iteration.

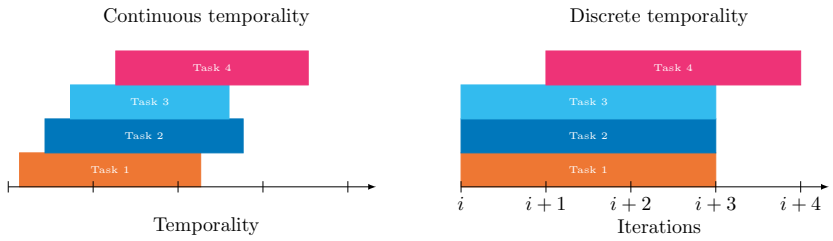

Fig. 3. Temporality: the figure on the left represents a continuous temporality and the one on the right a discrete temporality. Unlike continuous temporality, the discrete case includes tasks with identical execution times (3 cycles) and tasks no.1, 2 and 3 start and end at the same time. Discretization implies a standardization of execution times and simultaneity .

The new tasks generated are stored in a queue (a sorting by decreasing priority then by the closest deadline is carried out at each iteration) and processed sequentially. The decision-making process used depends on the solution.

4 Method definition

4.1 Market-based approaches

For this method, we consider two approaches: one with and one without preemption. They share the same decision making process, only the scope of the transfer mechanism differs.

Decision process
First, each agent tries to run their tasks locally as long as their resources allow it. Then, it tries to transfer the non-executed tasks to other robots. And finally, it postpones the remaining tasks that have not found a buyer. A complete diagram of the decision process of the different algorithms used in this article can be found in Fig. 5. The transfer action uses an auction system to select the best receiver.

The auction system
When an agent wants to transfer a job, it uses the auction system mechanism. It then becomes an auctioneer and proposes selling a task to all other agents in its area. Any agent receiving the offer submits a bid whose evaluation process is described in Algo. 1. The preemptive method allows agents to stop the execution of certain tasks to accept a transfer. Preemption can only be done to the detriment of lower priority tasks. Stopped tasks can be restarted by the agent but will start from scratch (as penalty) and must respect their initial deadline. In the preemptive method the auctioneer can participate to its own auction, which consequently allows local preemption. It is worth mentioning that the centralized nature of auctions does not contradict the decentralized nature of the system. Indeed, this centralization is temporary and any agent can become an auctioneer. Therefore, there is not introduction of a single point of failure (SPOF).

4.2 DQN approaches

Decision process
Unlike market-based approaches, the system has complete freedom over it decision process. The only limit being the validity of the action taken. Indeed, the system cannot perform an impossible action like allocating more resources than owning. The decision process depends on the Q-values calculated during the learning. Moreover, no preemption is allowed for this approach so the system must learn to refrain from performing low priority tasks to ensure high-priority task execution.

Algorithm 1: Bid valuation algorithm
(In red, steps specific to the pre-emptive approach)

Data: T, the auctioned task
A, the list of active tasks on the robot
A_i, the list of active tasks with a priority i
R_{cpu}, the amount of free CPU on the robot
R_{mem}, the amount of free memory on the robot
Function : $cpu(i)$, returns the CPU required by task i
$mem(i)$, returns the memory required by task i
Result: Bid, The bid valuation
if $R_{cpu} \geqslant cpu(T)$ *and* $R_{mem} \geqslant mem(T)$ **then**
  $Bid \leftarrow R_{cpu} + R_{mem}$;
else if $priority(T) = 1$ *and* $R_{cpu} + \sum_{i \in A \setminus A_1} cpu(i) \geqslant cpu(T)$ *and*
$R_{mem} + \sum_{i \in A \setminus A_1} mem(i) \geqslant mem(T)$ **then**
  $Bid \leftarrow R_{cpu} + R_{mem} - \sum_{i \in A_1} (cpu(i) + mem(i))$
else if $priority(T) = 2$ *and* $R_{cpu} + \sum_{i \in A_3} cpu(i) \geqslant cpu(T)$ *and*
$R_{mem} + \sum_{i \in A_3} mem(i) \geqslant mem(T)$ **then**
  $Bid \leftarrow R_{cpu} + R_{mem} - \sum_{i \in A \setminus A_3} (cpu(i) + mem(i))$
else
  $Bid \leftarrow -\rho$;
end

As mentioned earlier, our RL method relies on a DQN to predict Q-values. These Q-values determine which of the three actions (run, transfer or postpone) to choose in the current state. The architecture of our network comprises an input layer of 11 neurons, three hidden layers of 32 neurons each and an output layer of 3 neurons as described by Fig. 4. The input layer brings two types of information. The first one is the state of the agent and consists of six neurons and carry information such as the robot load, area, etc. The second one, composed of five neurons, delivers information relative to the task to allocate such as its computing requirement, priority, deadline, etc.

In this approach, there is no explicit system for selecting the receiver robot for the transferred task. As a result, the process differs from the auction approach.

DQN and transfer

The transfer action causes the release of a transfer proposal to other agents in the area. Each concerned agent uses its DQN to choose whether to accept the task or not. If only one agent responds favourably, it will execute the task at the

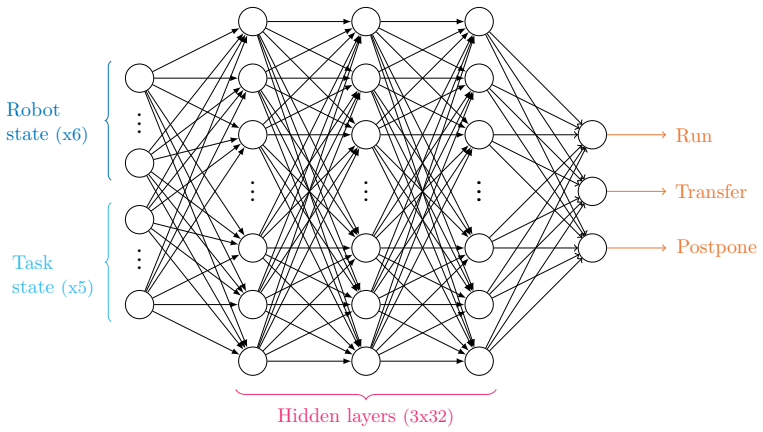

Fig. 4. Schematic view of our DQN architecture: the DQN is composed of an input layer of 11 neurons, three hidden layers of 32 neurons each and an output layer of 3 neurons

next cycle with the penalties due to the transfer. If several favourable answers exist, the agent is randomly selected regardless of the relevance of this choice. Agents must learn to properly evaluate the appropriate response to a transfer request. Finally, the task is postponed if no favourable response is sent to the auctioneer.

Another key factor of a DQN method is its reward system, which fully conditions the learning quality and its relevance.

The reward system

Our system has two objectives: the completion of a maximum of tasks and the respect of priorities. Moreover, the total freedom of this approach implies a third objective: the limitation of the use of the transfer in order to limit communication between the agents. Our system rewards differently if the action leads to performing a task (action 0 or action 1 successfully) or not.

$$r(s,a) = \begin{cases} V_i, & \text{if performed.} \\ \dfrac{-V_i \times p}{L}, & \text{otherwise.} \end{cases} \tag{7}$$

with:

- i, p, L : the priority, the number of postponements and the laxity of the task, respectively. These values are obtained from s.
- $V = [\alpha_1, \alpha_2, \alpha_3]$: reward based on priority

The values of α_1, α_2, α_3, β have been determined by intensive successive simulations and are 1, 0.3, 0.05, -0.2 , respectively. We will now present the other parameters of our experiments.

5 Experimental conditions

5.1 Experimentation set up

The simulator is designed to explore the parameters and test the relevancy of our solutions before a deployment on real mobile robots.

Emulator set up
Our emulator is coded in python 3.7.3. The learning and deployment part of DNN rely on the widely used Tensorflow (with Keras), open source ML framework. Learning methods based on DNN require intensive computing. To attest our method viability viability in real conditions, we tested the inference and learning times on an embedded architecture adapted to mobile robots. The Jetson TX2 is a power-efficient ($< 15W$) computing device suitable for Embedded AI. Since it is very popular on mobile robot, it will serve as a reference. As shown in Table 2, results attest our reinforcement approach is fully compatible with the computing power available on a robotic architecture.

Table 2. Inference and learning time by architecture

Configuration	Emulator	Nvidia Jetson TX2
Components	Intel Xeon Silver 4114 (x2) Nvidia GTX 1080 TI	ARM Cortex-A57 Nvidia Denver 2 GPU Pascal
Hardware	20 cores at 2.2 GHz 64 GB of DDR4 3584 cores CUDA 11 GB of GDDR5X	4 cores at 2 GHz+ 2 cores at 2Ghz 8 GB of LPDDR4 256 cores CUDA
Inference	2.3ms	4.8ms
Experience replay (batch of 24)	215ms	673ms

Reference performances
We introduce lower and upper bounds in order to get comparison points for our three methods: DQN, market based ('Auction') and market-based with preemption ('P-Auction').

– The lower bound: a purely decentralized and local approach acts as a lower bound ('L-Bound'). There is no communication between the agents. Each robot acts independently of the others. All other approaches must be superior because otherwise they are counter-productive.
– The upper bound: As an upper bound ('U-Bound'), we logically chose a centralized approach with preemption. Indeed, because of its overall knowledge of the state of the system, the centralized vision offers optimal performance. All decentralized systems must aim to achieve such performances. Since our system is spread over multiple areas, our centralization is done by area. Each of them has a central agent independent of the MRS and equivalent to a local cloud. At each iteration, the new tasks to be assigned to the area agents are, instead, assigned to the central agent of the corresponding area. Then, the central agent distributes the tasks by favouring the load balancing among all available agents.

All the decision-making processes are illustrated in Fig. 5. The terms "DQN", "Market", "Market-P," L-Bound "and" U-Bound " refer to the solutions: DQN, market without preemption, market with preemption, lower bound and upper bound respectively.

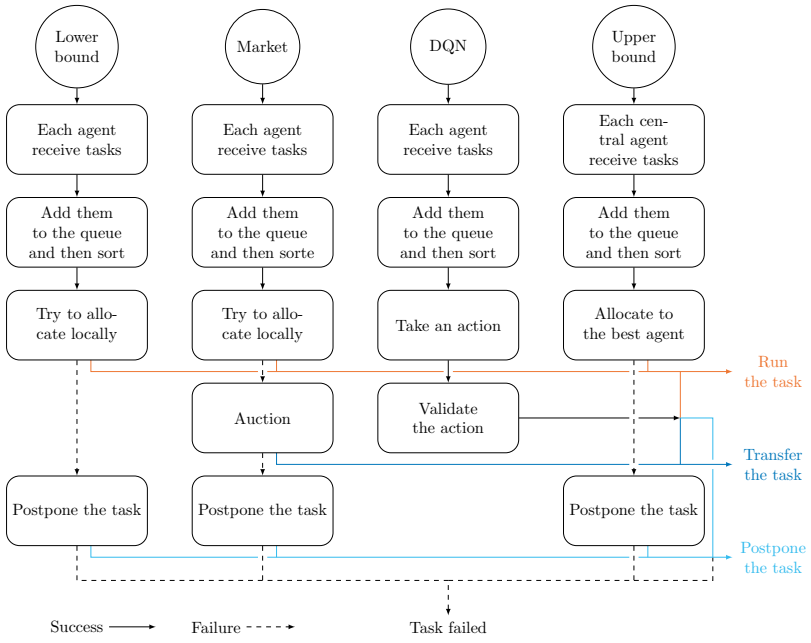

Fig. 5. Decision process diagram for the different methods

Experimentation configurations
The large number of possible configurations led us to restrict our study to a set of configurations where some parameters are fixed. These parameters were determined by intensive successive tests and are described in Tab. 3:

Table 3. DQN and environment settings: values in orange are those by default but are subject to change.

DQN settings	Value	Environment settings	Value
γ	0.5	Robots number	7
ε *decay*	0.995	Areas number	3
α	10	Maximum postpone	4
c	40	Execution time	4
Memory size	256	Number of priorities	3
batch size	24	Weighting (priorities choice)	(1, 1, 1)
$V = [\alpha_1, \alpha_2, \alpha_3]$	$[1, 0.35, 0.05]$		

The results presented for the DQN solution are those obtained after pre-training on a different mission lasting 300 iterations. No learning takes place during our tests to considerably accelerate the simulations (allowing to average the results).

5.2 Tasks configuration

Tasks assignment
Tasks appear over time and are randomly assigned to a robot. We used three distribution shown in Fig. 6 systems (repeated every 20 iterations) to define the task number by iteration :

1. Distribution 1: a serrated distribution with a minimum of 3 tasks and a maximum of 13 tasks. This is the default configuration and its close to the one used in [1].
2. Distribution 2 : a tiered distribution with a low tier having 5 tasks and a high tier with 11 tasks.
3. Distribution 3: a more complex distribution where each robot receives a task every two iterations and where some robots receive additional tasks. The distribution of additional tasks is as follows:
   - A robot (chosen randomly at each iteration) receives three more tasks in iterations 1 to 5 and 11 to 15.
   - Two robots (chosen randomly at each iteration) receives three more tasks in iterations 6 to 10 and 16 to 20.

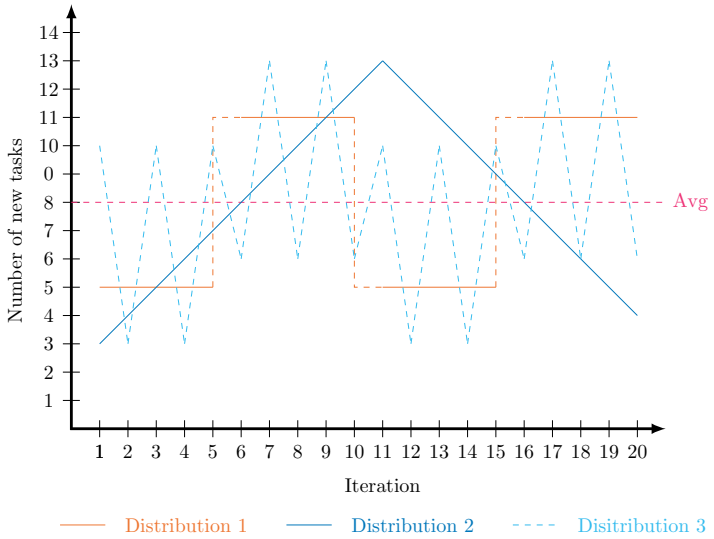

Fig. 6. Number of tasks allocated at each iteration based on the distribution used. The distributions are periodic and therefore repeated every 20 iterations until the end of the simulation.

The average number of assigned tasks per iteration is eight, which according to experiences and configurations seems to be the sweet spots between overload and complete system saturation. During the experiments, the approaches evolve in parallel in a strictly identical environment (the assignment of tasks is identical).

Data set

Upon its creation, a task receives a type defining the prerequisites necessary for its execution. This type is randomly and uniformly selected among those of the task set used. We use three sets of tasks :

1. The first task set is a condensed version of the one used in [1]. The sum of the prerequisites is always equal to 40 % 4
2. The second is the one used in [1] where large disparities exist between types 5.
3. the third is a imbalance version of the second set where types requiring more memory than CPU have been removed to incorporate some imbalance 6.

Area assignment

Our experiment includes an area system. In order to evaluate its impact on the system and more particularly on the transfer mechanism, we have defined three area configurations:

Table 4. Data set no.1

Tasks								
Type	CPU (%)	RAM (%)	Type	CPU (%)	RAM (%)	Type	CPU (%)	RAM (%)
1	20	20	2	25	15	3	15	25
4	35	05	5	30	10	6	10	30
7	05	35						

Table 5. Data set no.2

Tasks								
Type	CPU (%)	RAM (%)	Type	CPU (%)	RAM (%)	Type	CPU (%)	RAM (%)
1	5	5	9	15	10	16	10	15
2	10	10	10	25	15	17	15	25
3	15	15	11	30	5	18	5	30
4	20	20	12	30	20	19	20	30
5	25	25	13	35	20	20	20	35
6	30	30	14	35	25	21	25	35
7	35	35	15	40	10	22	10	40
8	40	40						

Table 6. Data set no.3

Tasks								
Type	CPU (%)	RAM (%)	Type	CPU (%)	RAM (%)	Type	CPU (%)	RAM (%)
1	5	5	6	30	30	11	30	5
2	10	10	7	35	35	12	30	20
3	15	15	8	40	40	13	35	20
4	20	20	9	15	10	14	35	25
5	25	25	10	25	15	15	40	10

1. a single area
2. two areas with a probabilities of 0.5 for each
3. three areas with the probabilities $\frac{1}{2}$, $\frac{1}{3}$ and $\frac{1}{6}$ for area 1, 2 and 3 respectively. This is the default configuration.

All robots start the mission in the first area. Every twenty iterations, each of robot is assigned to an area according to the configuration probabilities.

We now present the results of our experiments obtained with data-set described in Table 4 (unless otherwise stated).

6 Results

The results presented in this section are an average of the results obtained on 100 simulations of 1000 iterations each.

6.1 Overall performances

These simulations are intended to answer questions raised by the use of these approaches. The first question is:

- How do these strategies affect the system's capacity to complete its tasks?

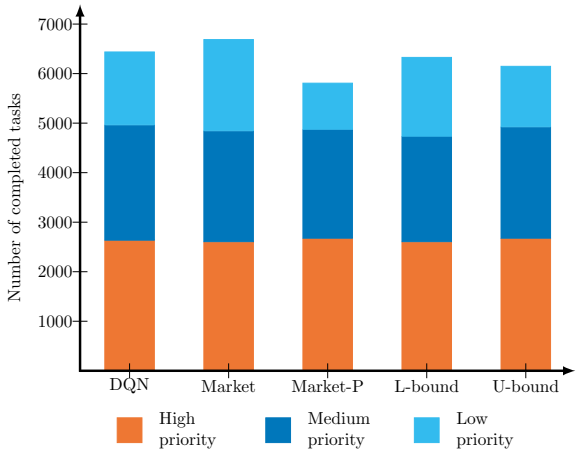

Fig. 7. Completion rate for each solution in the default configuration. The orange, blue and cyan parts represent the tasks of priority 1, 2 and 3, respectively.

Completion rate

Fig. 7, 8 show the number of tasks completed and missed respectively, according to their priorities for each approach. We can already see a strong difference between the two market approaches with a non-preemptive approach offering the best completion rate and the preemptive the worst. In the middle, we find the other approaches offering similar completion rate with the DQN being slightly better).

With an upper bound completing fewer tasks than the lower bound and market-p showing the worst results, the high cost of the preemption mechanism is obvious.

- Are preemptive solutions able to show better performance with respect of critical tasks?

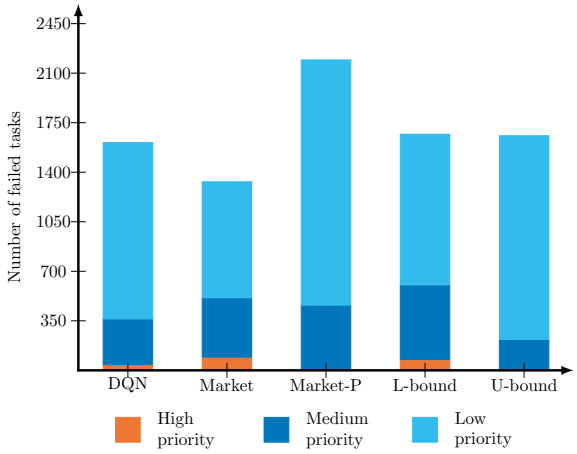

Fig. 8. Failure rate for each solution in the default configuration. The orange, blue and cyan parts represent the tasks of priority 1, 2 and 3, respectively.

Criticality management

Preemptive approaches are effective in fulfilling criticality with a zero failure rates for *1-priority* tasks, as shown in Fig. 8. The DQN approach is likely to provide better results than non-preemptive approaches for high priority task. It even outperforms the market-P approach for medium priority tasks This shows that this solution refrains from allocating as soon as possible and retains some scope for higher priority tasks. However, it fails to ensure all *1-priority* tasks.

We see a need for a trade-off. The system cannot both complete a maximum number of tasks and effectively maximize the completion of high priority tasks. The relevance of an approach therefore depends on the weighting given to the two objectives.

□ Can these results be explained by the load distribution ?

Load balancing

The ability of a solution to maximize the use of available resources is a good performance indicator. Fig. 9 shows the average load distribution of a robot during 1000 iterations. The cyan amount represents the resources spent on processing the transfer overhead. Distributed approaches show a strong disparity in the resources spent in overhead. The market-P uses the transfer mechanism the most allowing it to achieve its perfect 1-priority task completion rate, but at the expense of the resources spent on tasks(resulting on a weak overall completion rate). The market approach also spends an important part of resources in overhead allowing it to perform more tasks than the L-bound approach leading

to offer a much better overall completion rate. Compared to the market and market-P approaches, the DQN approach is the most moderate with limited transfer use. The latter is only used for high priority tasks and therefore does not result in a significant improvement in the overall completion rate.

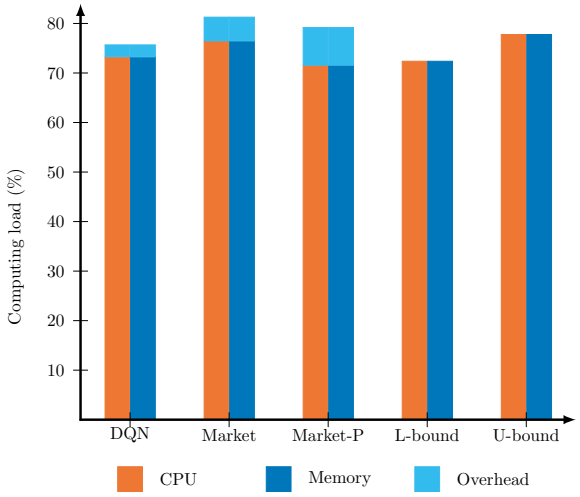

Fig. 9. Caption

Distributed systems manage to distribute the load by using the transfer mechanism.

☐ How often and effectively do market-based and DQN approaches use the transfer mechanism?

6.2 Quality of the distribution

Transfer and communication

Fig. 10 shows shows the average number of transfer requests depending on the solution and the area configuration. It appears that the DQN-type approach has more control over the transfer mechanism. Indeed, the success/failure rate is clearly more advantageous than for market-based approaches. As expected, more advantageous zone configurations (with fewer zones) offer better transfer success rates. An important observation is that the DQN solution seems to perceive this better success chances. As a result, it tries to transfer more often since it is less likely to be penalized. On the one hand, we manage to limit the number of transfers via our reward model as desired. On the other hand, we fail to

eliminate unsuccessful transfer requests. This is explained by the fact that the success of the transfer action does not depend solely on the transmitting agent. Although unable to prevent unsuccessful transfers, the DQN approach remains significantly more communication efficient than the market solutions, especially if we take into account the DQN solution's transfer mechanism minimalism .

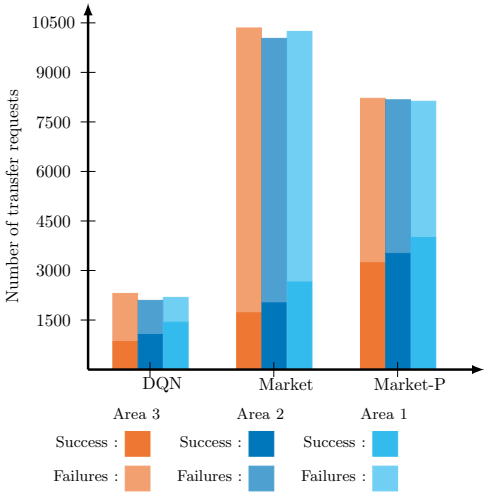

Fig. 10. Transfer mechanism success and failure rate in the default configuration: the orange, blue and cyan parts represent are configurations 3, 2 and 1 respectively.

The transfer's efficiency depends directly on the number of potential recipients.

☐ What happens to the performances if we change the area configuration to make transfer easier?

Impact of areas Tab. 7 compiles the performance of the different solutions according to the used zone configuration. As observed previously, reducing the area number allows agents to find new sales opportunities. This results in an increase in the number of successful transfers as well as a decrease in failures for the distributed approaches. This increase in the number of successful transfers leads to a higher overall success rate for the concerned solutions. The U-bound also benefits from these changes since it is centralized by area.

So far, the results attest the great adaptability of the DQN approach which remains moderate in order to perform a maximum of high and medium priority tasks, limit overhead and maintain an good overall completion rate

☐ What happens if we change the method to distributing the new tasks?

Table 7. Performance of the different solutions on the default configuration depending on the area configuration: the slight differences completed tasks totals can be explained by the presence of postponed or running tasks

Solution	Area	Completed			Failed			Load (%)		Overhead	
		P-1	P-2	P-3	P-1	P-2	P-3	CPU	MEM	CPU	MEM
L-bound	NC	2591	2129.5	1598.1	66.7	533.7	1068.5	72.3	72.3	NC	NC
	3	2589.6	2234.4	1842	74.4	416	820	76.3	76.3	4.8	4.8
Market	2	2586.7	2263.8	1869.8	69.8	395.2	790.8	77	77	5.8	5.8
	1	2601.1	2325.8	1910.3	46.9	329	762.9	78.3	78.3	7.5	7.6
	3	2260.8	2195.6	930.7	0.1	455.5	1731.2	71.5	71.4	7.6	7.6
Market-P	2	2657.8	2251.7	940.5	0	407.5	1720.6	71.8	71.9	8.8	8.8
	1	2655.1	2379.5	990	0	276.2	1682.8	73.1	73.1	10.8	10.8
	3	2624.9	2332.1	1413.2	35.3	321.1	1245.6	73.1	73.1	2.4	2.4
DQN	2	2635.7	2333.7	1430.4	18.7	242.4	1230.6	74.1	74.1	3	3
	1	2648.4	2349.3	1488	6.9	197.1	1182.2	75.4	75.4	4.1	4.1
	3	2665.4	2439.4	1120.5	0	214	1140.7	77.7	77.7	NC	NC
U-bound	2	2659	2523.3	1282.1	0	137.4	1378.2	78.9	78.9	NC	NC
	1	2654.4	2632.2	1470.1	0	25.6	1201.5	81.1	81.1	NC	NC

6.3 Distribution impact

Distribution by tier

The distribution by tier consists of two tiers of five and eleven tasks respectively and should lead to less disparity. The results obtained, for this configuration distribution, by the different configurations according to the area configuration are listed in Tab.8. We can see that this distribution is easier since the L-bound shows better performance with a better load (74.2 instead of 72.3) and a better completion rate of high priority tasks (51.6 vs 66.7). And indeed, the more concentrated nature of this distribution also allows other solutions to display better performance. If the DQN approach is no exception and offers better results, this increase is comparatively less than the other solutions. This is explained by its reluctance to use the transfer mechanism leading to its inability to take full advantage of the situation.

More complex distribution

The results obtained by the different solutions when the tasks generations is carried out with the distribution configuration no.3 are presented in Tab 9. Unlike the previous distribution, this one is slightly harder as the results obtained by L-bound attest. We should logically expect lower results also for the other solutions, but this is absolutely not the case and the results presented are even better. Indeed, the periodic and staggered nature of this distribution gives enough time to overloaded robots in order to transfer their tasks as attested by the significant

Table 8. Performance of the different solutions on the distribution configuration no.2 and depending on the area configuration

Solution	Area	Completed			Failed			Load (%)		Overhead	
		P-1	P-2	P-3	P-1	P-2	P-3	CPU	MEM	CPU	MEM
L-bound	NC	2595.5	2199.4	1680.3	51.6	446.8	987.8	74.2	74.2	NC	NC
	3	2595	2346.2	1942.7	55	313.6	708.2	78.9	78.9	5.4	5.4
Market	2	2601.5	2357.3	2007.2	46.2	288.5	660	79.7	79.7	6.4	6.4
	1	2626.8	2436.9	2054.4	24.7	218.8	598.1	81.5	81.5	8.6	8.6
	3	2650.2	2364.6	928.2	0	295.7	1723.1	74	74	8.2	8.2
Market-P	2	2647.1	2398.2	954.8	0	247.9	1713.2	74.4	74.4	9.5	9.5
	1	2651.6	2524.5	1017.5	0	131.4	1635.8	75.9	75.9	11.9	11.9
	3	2626.8	2321.3	1694.1	28.7	248.2	991.7	76.9	76.9	2.7	2.7
DQN	2	2639.7	2336.6	1692	12.3	160.4	1026.8	77.3	77.3	3.5	3.5
	1	2643.7	2470.5	1615.9	8.9	114	1041.3	77.9	77.9	5.1	5.1
	3	2651.9	2531.6	1398.8	0	130	1252.4	81	81	NC	NC
U-bound	2	2648.8	2575.2	1523.6	0	72.9	1144.9	82.5	82.5	NC	NC
	1	2653.7	2648.2	1856.8	0	9.8	802.1	85.9	85.9	NC	NC

amount of overhead. As seen with the previous distribution, the DQN reluctance to transfer prevents it from showing significantly better performance.

Table 9. Performance of the different solutions on the distribution configuration no.3 and depending on the area configuration

Solution	Area	Completed			Failed			Load		Overhead	
		P-1	P-2	P-3	P-1	P-2	P-3	CPU	MEM	CPU	MEM
L-bound	NC	2587.4	2152.2	1534.9	62.34	519.4	1118.6	71.9	71.9	NC	NC
	3	2611.9	2347.3	1924.9	45.8	311.8	726.1	78.7	78.7	5.6	5.6
Market	2	2617.5	2379.7	1987.5	38	279	665.9	79.9	79.9	6.7	6.7
	1	2631.2	2477.4	2080.9	14	186.6	576.4	82.3	82.3	9.1	9.1
	3	2657.5	2377.8	939.5	0.7	281.8	1711.8	74	74	8.4	8.4
Market-P	2	2655.4	2446	937	0	212.7	1717	74.7	74.7	9.9	9.9
	1	2645.3	2585.8	995.1	0	78.1	1662.3	76.6	76.6	12.7	12.7
	3	2632.4	2429.4	1381.4	27	234.3	1270.5	73.8	73.8	2.7	2.7
DQN	2	2639.1	2499.6	1395.8	9.7	162	1256	74.7	74.7	3.5	3.5
	1	2638.3	2578.7	1622.4	10.1	93.7	1036.5	77.9	77.9	6.3	6.3
	3	2659.2	2529	1447.7	0	131.9	1204	81.2	81.2	NC	NC
U-bound	2	2657.4	2598.8	1598.9	0	62.2	1064.4	83	83	NC	NC
	1	2658.9	2578.7	1622.4	0	2.2	557.3	87.4	87.4	NC	NC

The DQN approach struggles to take advantage of a stable situation.

- What happens if we introduce some imbalance to challenge the different solutions?

6.4 Impact of the task set and imbalance

Disparity

Fig. 11 and 12 show the completion and failure rates obtained with task set no.2 on the distribution no.1 with the area configuration no.3. All solutions except DQN show lower results. No more approach can manage to execute all high priority tasks since U-bound and market-P failed 2.4 and 3.7 high priority tasks respectively.

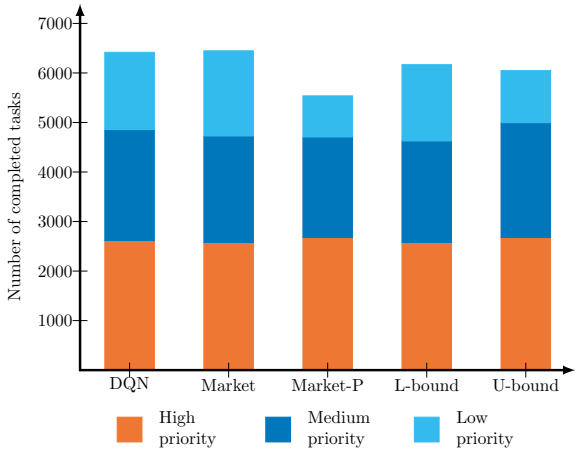

Fig. 11. Completion rate for each solution with the task set no.2. The orange, blue and cyan parts represent the tasks of priority 1, 2 and 3, respectively.

The Fig. 13 shows the average load of each solution with task set no.2. First, we can observe an overall increase in the computing load. This increase comes from the task set which (on average) requires more computing resources, which explains the performance decrease. Second, the DQN solution manages to take advantage of the task set disparity by showing a significantly better load leading to its good performance.

These results can be explained by the presence, in the task set, of tasks requiring little or a lot of resources (unlike the previous game). As a task is valued only according to its completion or / and its priority, but not according to its computational cost, the DQN approach can set up a strategy. It refrains from performing tasks requiring too many resources in order to maximize the overall number of tasks performed. This explains the slightly lower performance for high priority tasks. The result is an efficient system that has taken advantage of the situation. Of course, during a real deployment, it is impossible to skip the tasks requiring high computational resources. But this attests of the adaptive character of RL approaches which can adapt unlike classical approaches.

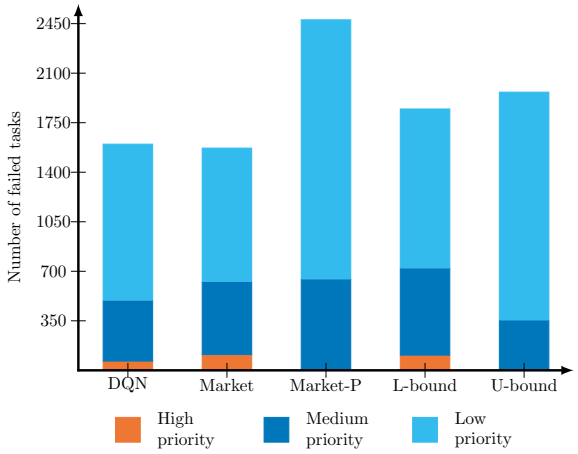

Fig. 12. Failure rate for each solution with the task set no.2. The orange, blue and cyan parts represent the tasks of priority 1, 2 and 3, respectively.

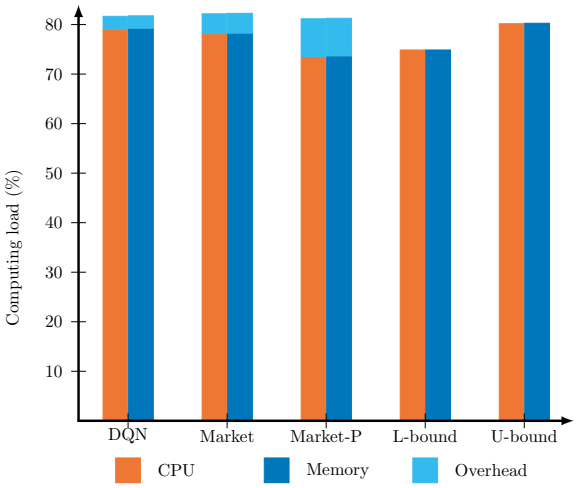

Fig. 13. Average load of the different solutions obtained with the tasks set no.2: the orange, blue and cyan parts represent the cpu load, memory load and the resources spent in overhead respectively.

Imbalance

In order to model an imbalance, we use the task set no.3 that favours one of the metrics, the CPU (of course, the results are identical if we choose to favour the memory). Fig. 14 and 15 show the completion and failure rates obtained

with task set no.3. As expected, the strong imbalance causes poor performance for all solutions. The preemptive solutions are particularly affected since the execution of a strongly unbalanced task can lead to the eviction of balanced tasks increasing the imbalance. Once again (but in a more moderate way), we notice that the DQN approach is able to adapt to the situation by offering a very good completion rate while maintaining a respect for priorities.

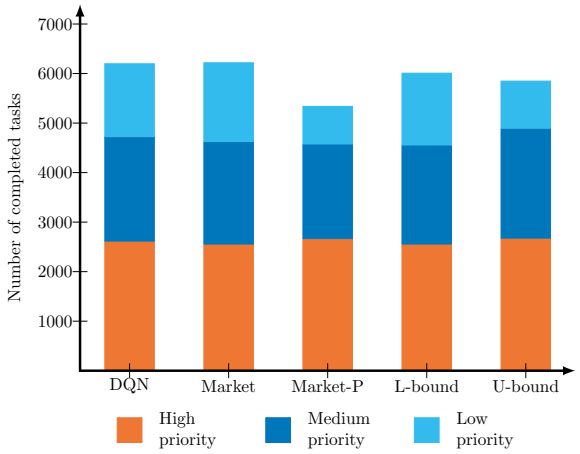

Fig. 14. Completion rate for each solution with the task set no.3. The orange, blue and cyan parts represent the tasks of priority 1, 2 and 3, respectively

Fig. 16 shows the average robots load for the different solutions when task set no.3 is used. . The great disparity in between resources type explains the poor performance since the system capacities cannot be fully mobilized. We can observe a better load balancing for the DQN solution. This attest of the presence of a strategy favoring balanced tasks

The DQN approach seems to be more relevant when the environment requires adaptability.

□ What happens if we introduce some imbalance into the priorities distribution?

6.5 Priority

In order to incorporate some imbalance in the priorities distribution, we have modified the weights assigned to them. The distribution used keep a weight of

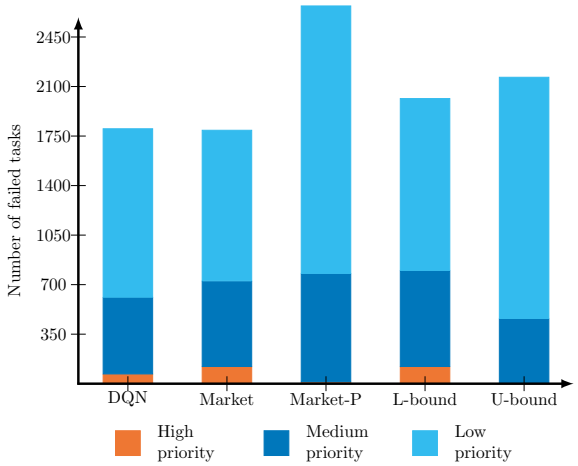

Fig. 15. Failure rate for each solution with the task set no.3. The orange, blue and cyan parts represent the tasks of priority 1, 2 and 3, respectively

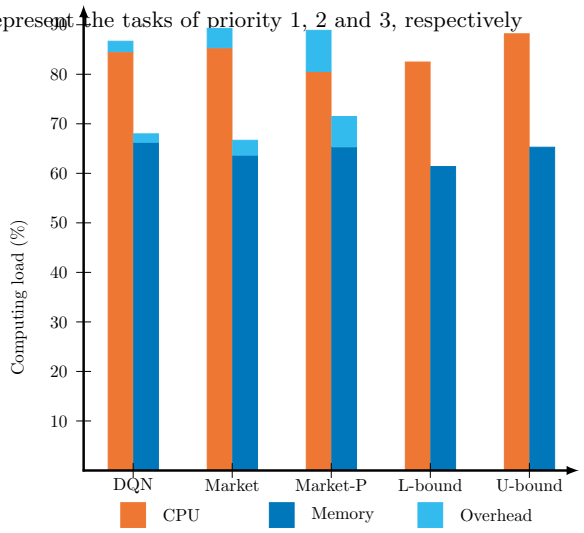

Fig. 16. Average load of the different solutions obtained with the tasks set no.3: the orange, blue and cyan parts represent the cpu load, memory load and the resources spent in overhead respectively.

1 for high and low priorities but give a weight of 0.5 for medium priority. As a result, robots are assigned twice as many high and low priority tasks than medium priority.

Fig. 17 and 18 show the completion and failure rates obtained with the new priorities distribution. On the one hand, there is not much change to be observed

in the classical approaches compared to the results observed with the uniform distribution. Except for non-preemptive approaches which see their high priority failure rate slightly increased, but the one for medium priority task decreased (logical consequence of the new distribution). On the other hand, the approach of the DQN solution seems different with a much better respect for priority at the cost of a significantly lower overall completion rate.

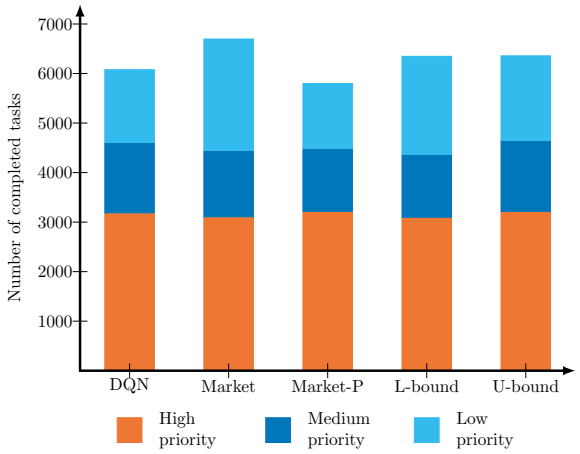

Fig. 17. Completion rate for each solution with with a non uniform priorities distribution. The orange, blue and cyan parts represent the tasks of priority 1, 2 and 3, respectively

Fig. 19 shows the average robots load for the different solutions when the priorities distribution is not uniform. As for the completion rates, the classic solutions load rates appear identical to those observed in the default configuration.

The DQN approach sets up a resource management strategy that is profoundly different from the one observed previously. On the one hand, this solution is even more restrictive of allocating its resources, which explains a lower payload rate than other approaches. On the other hand, it makes more use of the transfer mechanism resulting in an increase in the amount of overhead generated. These two changes describe the new strategy of this approach which focuses on high priority tasks taking advantage of their large number. Indeed, the transfer mechanisme becomes more interesting since high priority tasks brings more reward. In addition, restraining from allocating to execute a high priority task becomes more profitable since it will probably be done to the detriment of low priority tasks which bring little penalized in case of failures.

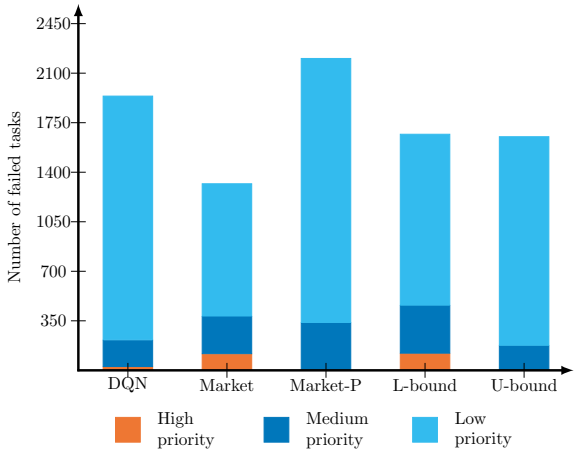

Fig. 18. Failure rate for each solution with a non uniform priorities distribution. The orange, blue and cyan parts represent the tasks of priority 1, 2 and 3, respectively

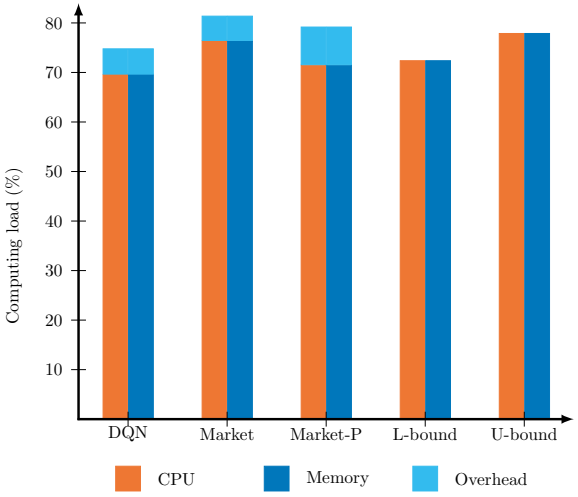

Fig. 19. Average load of the different solutions obtained with a non uniform priorities distribution: the orange, blue and cyan parts represent the cpu load, memory load and the resources spent in overhead respectively.

This choice to favor priority may seem extreme, but it results directly from the reward system designed. Without a proper score metric making the link

between the priority and the completion rate, we had to make an arbitrary choice concerning the values of the different priorities.

7 Conclusion and future work

In this article, we propose a comparative analysis of two types of approaches to solve the MRpTA problem that emerge to take advantage of shared computing resources within a MRS. In a distributed context, agents must spread the computational load through the transfer mechanism to compensate for local overheads. To complicate the problem, the transfer has a range limit and generates a penalty (20%).

As the complexity of the system is increasing, we have imposed a simple auction valuation algorithm to represent the impossibility of correctly valuing the bids of a real system. As far as RL is concerned, we have limited the size of the neural network architecture to make it compatible with mobile robot computing capabilities. In addition, we impose a simple reward scheme for the same reasons as auction valuation.

Despite these restrictions, the use of learning is relevant. Indeed, the tested solution is able to schedule tasks efficiently as evidenced by the good completion rates obtained while respecting a priority system. This solution manages to ensure a higher completion rate for medium priority tasks than other distributed approaches while offering performance close to the preemptive approach for high priority tasks. In addition, the transfer mechanism is controlled with a small failure rate and use adapted to the zone configuration limiting the MRS communications. In general, this approach offers excellent adaptability as its is able to define a strategy according to the environment in which it operates.

There are however some expected limitations. Indeed, while most of the objectives have been achieved, some have not been completely fulfilled. First, although respecting the priority system, the DQN approach has proved to be unable to ensure the execution of all high priority tasks. Second, despite a controlled transfer mechanism, it has not been used sufficiently with stable distributions. In addition, transfer failures remain present which can be explained by the totally distributed nature. Indeed, the success of the transfer action does not depend solely on the transferring robot.

Overall our conclusion to this study is that DQN appears as very relevant alternative. This first comparative approach to this MRpTA problem paves the way for the two following future works.The first one is to allow simultaneous / parallel decision-making on all tasks present in the robot's queue in order to overcome the limits of sequential decision-making and to offer a global vision to the learning agent. However, this raises many challenges related to state space and action space. The second one is to introduce a more accurate model for heterogeneous MPSOC (multi-core processors, GPU), a concrete simulation of robot movements and the use of real task sets.

References

1. Gautier, P., Laurent, J., Diguet, J.P.: Comparison of Market-based and DQN methods for Multi-Robot processing Task Allocation (MRpTA). In: 2020 Fourth IEEE International Conference on Robotic Computing (IRC). pp. 336–343. IEEE, Taichung, Taiwan (Nov 2020). https://doi.org/10.1109/IRC.2020.00060, https://ieeexplore.ieee.org/document/9287887/
2. Gerkey, B.P., Matari, M.J.: A Formal Analysis and Taxonomy of Task Allocation in Multi-Robot Systems. The International Journal of Robotics Research **23**(9), 939–954 (Sep 2004). https://doi.org/10.1177/0278364904045564, http://journals.sagepub.com/doi/10.1177/0278364904045564
3. Grandl, R., Ananthanarayanan, G., Kandula, S., Rao, S., Akella, A.: Multi-resource packing for cluster schedulers. In: Proceedings of the 2014 ACM conference on SIGCOMM - SIGCOMM '14. pp. 455–466. ACM Press, Chicago, Illinois, USA (2014). https://doi.org/10.1145/2619239.2626334, http://dl.acm.org/citation.cfm?doid=2619239.2626334
4. van Hasselt, H., Guez, A., Silver, D.: Deep Reinforcement Learning with Double Q-learning. arXiv:1509.06461 [cs] (Dec 2015), http://arxiv.org/abs/1509.06461, arXiv: 1509.06461
5. Hasselt, H.V.: Double Q-learning p. 9 (2010)
6. Jia, X., Meng, M.Q.: A survey and analysis of task allocation algorithms in multi-robot systems. In: 2013 IEEE International Conference on Robotics and Biomimetics (ROBIO). pp. 2280–2285 (Dec 2013). https://doi.org/10.1109/ROBIO.2013.6739809
7. Kehoe, B., Patil, S., Abbeel, P., Goldberg, K.: A Survey of Research on Cloud Robotics and Automation. IEEE Trans. Automat. Sci. Eng. **12**(2), 398–409 (Apr 2015). https://doi.org/10.1109/TASE.2014.2376492, http://ieeexplore.ieee.org/document/7006734/
8. Kingma, D.P., Ba, J.: Adam: A Method for Stochastic Optimization. arXiv:1412.6980 [cs] (Dec 2014), http://arxiv.org/abs/1412.6980, arXiv: 1412.6980
9. Koenig, S.: The Power of Sequential Single-Item Auctions for Agent Coordination p. 5 (2006)
10. Marjovi, A., Choobdar, S., Marques, L.: Robotic clusters: Multi-robot systems as computer clusters. Robotics and Autonomous Systems **60**(9), 1191–1204 (Sep 2012). https://doi.org/10.1016/j.robot.2012.05.007, https://linkinghub.elsevier.com/retrieve/pii/S0921889012000632
11. Mnih, V., Badia, A.P., Mirza, M., Graves, A., Lillicrap, T.P., Harley, T., Silver, D., Kavukcuoglu, K.: Asynchronous Methods for Deep Reinforcement Learning. arXiv:1602.01783 [cs] (Jun 2016), http://arxiv.org/abs/1602.01783, arXiv: 1602.01783
12. Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., Riedmiller, M.: Playing Atari with Deep Reinforcement Learning (Dec 2013), https://arxiv.org/abs/1312.5602v1
13. Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A.A., Veness, J., Bellemare, M.G., Graves, A., Riedmiller, M., Fidjeland, A.K., Ostrovski, G., Petersen, S., Beattie, C., Sadik, A., Antonoglou, I., King, H., Kumaran, D., Wierstra, D., Legg, S., Hassabis, D.: Human-level control through deep reinforcement learning. Nature **518**(7540), 529–533 (Feb 2015). https://doi.org/10.1038/nature14236, http://www.nature.com/articles/nature14236

14. Otte, M., Kuhlman, M., Sofge, D.: Multi-robot task allocation with auctions in harsh communication environments. In: 2017 International Symposium on Multi-Robot and Multi-Agent Systems (MRS). pp. 32–39 (Dec 2017). https://doi.org/10.1109/MRS.2017.8250928
15. Wan, J., Tang, S., Yan, H., Li, D., Wang, S., Vasilakos, A.V.: Cloud Robotics: Current Status and Open Issues. IEEE Access pp. 1–1 (2016). https://doi.org/10.1109/ACCESS.2016.2574979, http://ieeexplore.ieee.org/document/7482658/
16. Wang, Z., Li, M., Li, J., Cao, J., Wang, H.: A task allocation algorithm based on market mechanism for multiple robot systems. In: 2016 IEEE International Conference on Real-time Computing and Robotics (RCAR). pp. 150–155 (Jun 2016). https://doi.org/10.1109/RCAR.2016.7784017
17. Yahya, A., Li, A., Kalakrishnan, M., Chebotar, Y., Levine, S.: Collective robot reinforcement learning with distributed asynchronous guided policy search. In: 2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS). pp. 79–86. IEEE, Vancouver, BC (Sep 2017). https://doi.org/10.1109/IROS.2017.8202141, http://ieeexplore.ieee.org/document/8202141/