RESEARCH PROJECT IN MECHATRONICS ENGINEERING

Multi-agent Collaborative Control of a Mobile Robot Fleet using Reinforcement Learning

Munirah binte Mohd Mahadi

Supervisor: Dr Yuqian Lu

Department of Mechanical Engineering
The University of Auckland

19 May 2021

Multi-agent Collaborative Control of a Mobile Robot Fleet using Reinforcement Learning

Munirah binte Mohd Mahadi

ABSTRACT

Factories of the future need more flexible material transport systems that can dynamically schedule transport tasks under uncertain conditions. Popular methods to solve this task allocation problem include a range of algorithms that apply reinforcement learning. This research aims to develop multi-agent reinforcement learning algorithms for dispatching transport tasks in an autonomous mobile robot fleet. A Deep Q Neural Network was used in conjunction with a multi-agent system to realise the model. The novel "look-ahead" parameter was introduced to encourage the most efficient task allocation scheme and prevent the model from behaving similar to a first-in, first-out (FIFO) model. Extensive simulations were executed on a range of devices to ensure that the model was viable across varying hardware and software specifications. Each simulation consisted of a set number of agents and tasks, paired with parameters varied to determine the viability of the look-ahead parameter and validate the performance of the model under different environments. To further validate the model, experiments were conducted on a physical mobile robot fleet consisting of two TurtleBots. The results show that the model showed tremendous improvement when comparing the original performance to the trained model at the end of each simulation. Furthermore, the results also indicated that the "look-ahead" parameter improved performance regarding the distance travelled. However, this caused delays in the overall execution due to the additional calculations. These results suggest that the proposed model is viable and has the potential to increase the performance of mobile robot fleets in real applications. Modifications to the neural network and overall model could allow for further improvements and outperform traditional methods.

DECLARATION

Student

I hereby declare that:

1. This report is the result of the research project work carried out by myself under the guidance of my supervisor (see cover page) in the 2020 academic year at the Department of Mechanical Engineering, Faculty of Engineering, University of Auckland.
2. This report is not the outcome of work done previously.
3. This report is not the outcome of work done in collaboration, except that with a potential project sponsor (if any) as stated in the text.
4. This report is not the same as any report, thesis, conference article or journal paper, or any other publication or unpublished work in any format.

In the case of a continuing project, please state clearly what has been developed during the project and what was available from previous year(s):

Signature:

Date:

Table of Contents

List of Figures

List of Tables

List of Algorithms

Acknowledgements

I would like to thank and acknowledge my supervisor **Dr Yuqian Lu** for providing guidance and help throughout the project.

I would like to thank my friends and family for their unwavering support and motivation. I would especially like to thank my friends, **Vandna Patel** and **Dazzle Johnson**, for providing emotional and academic support throughout the course of this project.

I would also like to thank honours students **Finn O'Brien** and **Blake Carter Wagstaff** for their assistance with the AMR system.

I would also like to thank lab technician **Ehsan Amiri Tehranizadeh** for providing technical help and resources throughout the course of the project.

Glossary of Terms

Keras	An open-source library that interfaces artificial neural networks with Python
Tensorflow	An open-source library designed for machine learning purposes
Task	A transportation order, that needs to be transported from a start position to its end position
Dispatch	The assignment of tasks to agents
Individual queue, N	The amount of tasks that an agent can hold in its own memory
Look-ahead, K	The number of tasks assessed while bidding
Leadtime	The amount of time between a task's creation and its completion

Abbreviations

ROS	Robot operating system
HER	Hindsight Experience Replay
DQN	Deep Q Learning
A2C	Advantage actor critic
A3C	Asynchronous advantage actor critic
MARL	Multi-agent reinforcement learning
AMR	Autonomous mobile robot
SLAM	Simultaneous localisation and mapping
EMA	Exponential moving average

1. Introduction

Factories of the future need more flexible material transport systems that can dynamically schedule transport tasks under uncertain conditions. This concerns the task allocation problem, which can be described as the process of determining the most efficient allocation of a set of tasks to a set of agents for execution.

A significant challenge with this is that existing solutions can only provide a globally optimal solution when the environment is static and fully observable. These solutions, however, are only suitable for small scale environments, and in some cases, for single agents. Existing solutions that attempt to solve the task allocation problem for an environment with multiple agents have convergence issues. This is attributed to the dynamic environment associated with multiple agents learning in parallel. A wide range of reinforcement learning algorithms have been proposed and tested as a solution; however, these often require intensive computing resources, often unavailable on-board agents. Furthermore, factories of the future would consist of a team of heterogeneous agents. Since existing applications have focused on teams that consist of homogeneous agents, this added complexity should be accounted for increased flexibility and applications.

This research aims to develop a solution to the task allocation problem using multi-agent reinforcement learning that allows for heterogeneous agents within a single team while facilitating the convergence of the model to the optimal solution.

2. Literature Review

This literature review seeks to identify gaps in existing task allocation methods and approaches. These methods currently fall into one of two categories - auction-based or behaviour-based. Reinforcement learning as a solution for the task allocation problem is reviewed, and applications of various algorithms are assessed. This includes methods that employ value iteration and policy iteration, which are extended to multi-agent reinforcement learning.

2.1 Task Allocation

2.1.1 Auction-based

Auction-based approaches to task allocation are methods that utilise an auction, where the winner typically receives the available task. These approaches are very popular as it facilitates a distributed system whereby each agent can decide the value of taking a task independently. This is often taken advantage of, and the centralised controller can be removed from a system to increase scalability and reduce computational costs overall [1].

This approach has been successful in several research papers such as [1], [2], [3] and [4], where the performance met or exceeded the requirements of the respective systems.

In auctions, the typical process is that each robot assesses how valuable the task is and determines a bid based on that assessment. A cost function is commonly used to produce the bid. An auctioneer gathers the bids from all agents and determines which agent the task should be awarded. This can be either the highest or lowest bidder depending on how the system has been modelled [5].

[6] proposes a novel resource-based approach that varies from traditional bid generation approaches, which typically calculate the robot's bid assuming that it has sufficient resources required to complete the task by incorporating the robot's resources into the bid generation process. Compared to conventional approaches, the performance of the proposed approach was deemed superior as it minimised resource usage while minimising the time required to complete tasks. Furthermore, the proposed approach allowed for reduced communications between robots, further reducing the computational cost associated with the approach.

Cost functions play an essential role in auction-based approaches, as this cost determines the "winner" of each auction and determines which robot the task is allocated. Furthermore, the cost can be used in analysis to assess whether there are significant flaws in the system. The cost function also controls how the robots behave in the system, controlling whether they are selfish or cooperative. It also controls whether their bid is motivated by the overall success of the team [4].

A comparison of algorithms used for an auction-based task allocation system was conducted in [5]. The greedy algorithm, Hungarian algorithm, parallel auction algorithm, and an improved algorithm were implemented, and their performance was assessed.

The greedy algorithm heuristically solves the problem by choosing the local optimum, leading the system towards the global optimum. This algorithm, along with other heuristic algorithms, is typically the simplest valid option, as it takes the action that is considered to be the most valuable at any given time. In the task allocation system, this process is repeated for all the tasks that require assignment.

The Hungarian algorithm solves the problem with the assumption that there is a bipartite graph [5]. A bipartite graph is a graph where every edge in the graph joins a vertex in one set to a vertex in the other set, with no adjacent vertices in either set. The robots are considered one set, while the tasks are considered the other set. The algorithm looks to select a subgraph with the maximum number of edges to produce the optimal outcome.

The parallel auction algorithm was introduced by Zhang and Meng, where the auction

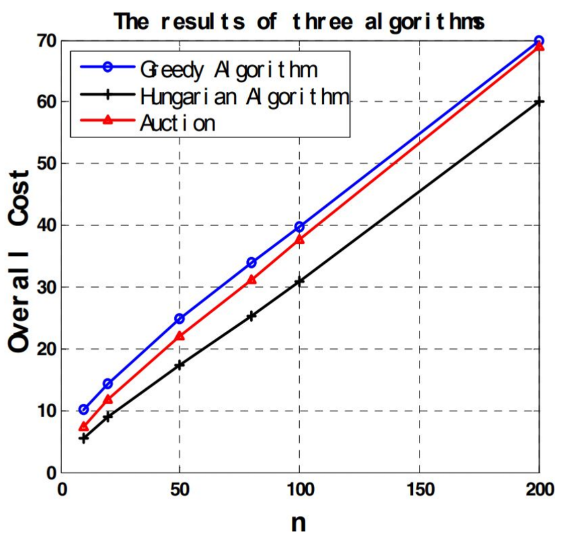


Fig. 1.The average overall costs of the three algorithms

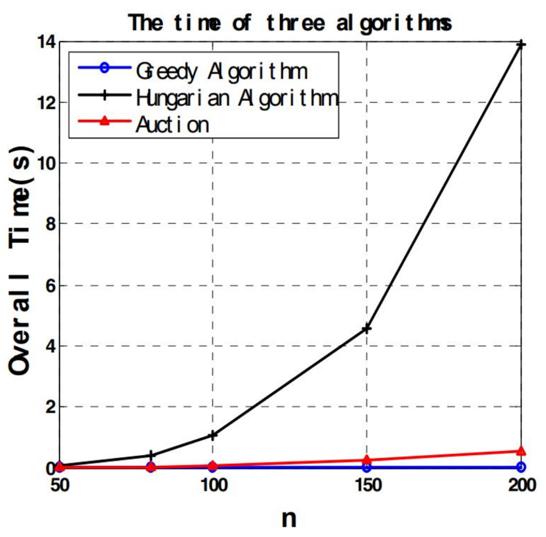


Fig. 2.The executive time of the three algorithms

Figure 1 Performance comparison of Greedy, Hungarian and Improved-parallel auction algorithms [5]

of tasks happens in parallel, and all robots are involved in this process. The overall structure of the algorithm is as follows; all robots bid on all available tasks, each of the robots that indicate the lowest cost are awarded their respective tasks. However, if a robot is the lowest cost for multiple tasks, the larger cost task auction is repeated. This entire process is repeated until each of the tasks has been assigned. The improved auction incorporates the auction time into the cost function, whereby an incremental function with a logarithmic base was implemented to optimise the overall time and costs of the system. Figure 1 shows the comparison of the time and cost associated with each algorithm [5]. Overall, the greedy algorithm was superior with regards to execution time in larger environments with larger dimensions. The Hungarian algorithm was able to determine the global optimum; however, its execution time in larger environments was inferior to the greedy algorithm. The improved auction showed advantages for systems that require a balance between the two objectives - execution time and overall performance.

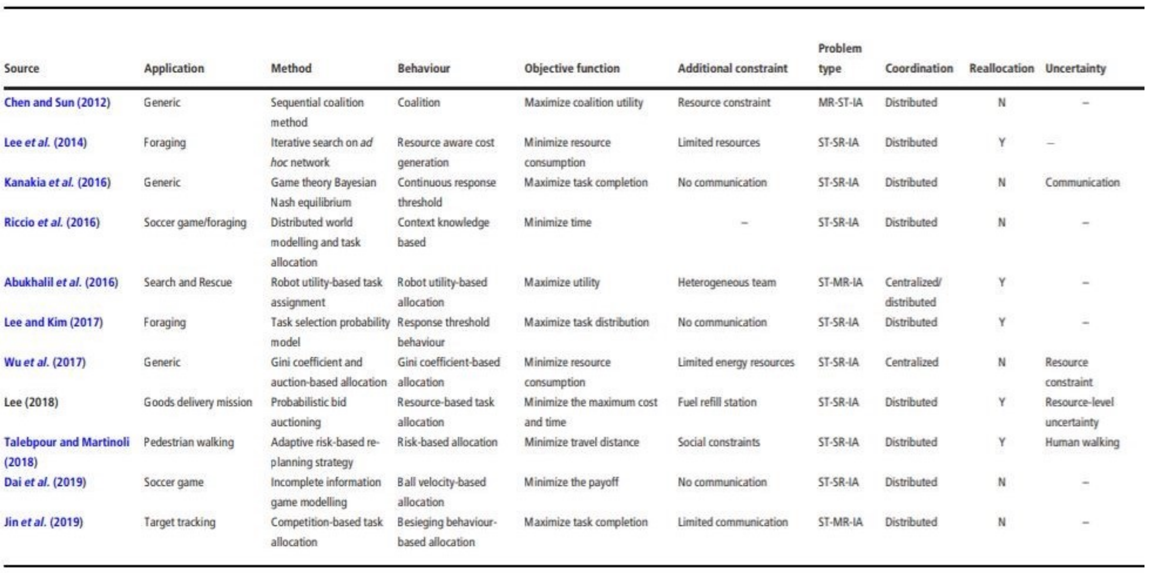
This shows that existing algorithms satisfy the needs of systems; however, there is significant room for improvement. Furthermore, while each method solves the task allocation problem, each approach has downfalls and shortcomings. There is evidence that these could be eliminated by combining techniques to achieve the advantages of multiple algorithms.

An application where improvements to the traditional algorithms can be found in [7], where an auction algorithm is modified to adapt to very dynamic environments. Cooperative hunting behaviour is achieved and it displays an experimentally proven improvement in performance through a 54% reduction in execution time. From this, it is evident that the traditional algorithms for task allocation are a good base algorithm to augment and modify for significant performance improvement.

2.1.2 Behaviour-based

Behaviour-based approaches to the task allocation problem are another popular approach that reacts strongly to the environment. There are typically different layers of control in task allocation problems - the higher-level behaviours such as task identification and communication between robots and the lower level behaviours such as navigation and obstacle avoidance. This approach is adaptable to problem-specific requirements from the ability to add to the robot's behaviours. Furthermore, this approach is suitable for both distributed and centralised systems, especially in a distributed system where global communication is not required. A system where the robots have weak communication would not be significantly disadvantaged with this approach. This is particularly suitable for applications where the environment is constantly changing, and the system needs to be able to adapt to the dynamic nature of the environment [8].

[9] is an application of a behaviour-based approach, where a fully distributed, behaviour-based system is implemented for a multi-robot task allocation problem. This study, in particular, uses ROS to assist the task allocation problem. The system was deemed successful based on the experimental results that indicated satisfactory performance for a team of robots. An analysis of behaviour-based approaches can be found in Figure 2 [8], where multiple applications and overall performance are summarised.



Source	Application	Method	Behaviour	Objective function	Additional constraint	Problem type	Coordination	Reallocation	Uncertainty
Chen and Sun (2012)	Generic	Sequential coalition method	Coalition	Maximize coalition utility	Resource constraint	MR-ST-IA	Distributed	N	–
Lee *et al.* (2014)	Foraging	Iterative search on *ad hoc* network	Resource aware cost generation	Minimize resource consumption	Limited resources	ST-SR-IA	Distributed	Y	–
Kanakia *et al.* (2016)	Generic	Game theory Bayesian Nash equilibrium	Continuous response threshold	Maximize task completion	No communication	ST-SR-IA	Distributed	N	Communication
Riccio *et al.* (2016)	Soccer game/foraging	Distributed world modelling and task allocation	Context knowledge based	Minimize time	–	ST-SR-IA	Distributed	N	–
Abukhalil *et al.* (2016)	Search and Rescue	Robot utility-based task assignment	Robot utility-based allocation	Maximize utility	Heterogeneous team	ST-MR-IA	Centralized/distributed	Y	–
Lee and Kim (2017)	Foraging	Task selection probability model	Response threshold behaviour	Maximize task distribution	No communication	ST-SR-IA	Distributed	Y	–
Wu *et al.* (2017)	Generic	Gini coefficient and auction-based allocation	Gini coefficient-based allocation	Minimize resource consumption	Limited energy resources	ST-SR-IA	Centralized	N	Resource constraint
Lee (2018)	Goods delivery mission	Probabilistic bid auctioning	Resource-based task allocation	Minimize the maximum cost and time	Fuel refill station	ST-SR-IA	Distributed	Y	Resource-level uncertainty
Talebpour and Martinoli (2018)	Pedestrian walking	Adaptive risk-based re-planning strategy	Risk-based allocation	Minimize travel distance	Social constraints	ST-SR-IA	Distributed	Y	Human walking
Dai *et al.* (2019)	Soccer game	Incomplete information game modelling	Ball velocity-based allocation	Minimize the payoff	No communication	ST-SR-IA	Distributed	N	–
Jin *et al.* (2019)	Target tracking	Competition-based task allocation	Besieging behaviour-based allocation	Maximize task completion	Limited communication	ST-MR-IA	Distributed	N	–

Figure 2 Summary of behaviour-based task allocation [8]

Behaviour-based approaches are also quite commonly used in situations where a controller is required for the system [10]. [11] uses an adaptive controller in conjunction with a system based on vacancy chains. The experimentation results show that the approach proposed in [11] is superior in situations where adaptation to changes in the environment is required, compared to other static approaches to task allocation. However, this architecture does not implement communication between the robots, which reduces its applicability for distributed systems where this communication is a necessity for full operation.

2.2 Reinforcement Learning

Reinforcement learning is a machine learning approach that allows an agent to explore behaviours autonomously. Through trial and error, an optimal behaviour is determined [12]. The environment is set up with time-steps over the entire horizon; at each time step, the agent observes its current state, which depends on its position in the environment and other factors that are determined by the design of the system.

Within reinforcement learning, there are different approaches to finding an optimal solution. These are classified into value iteration and policy iteration approaches that endeavour to reap the benefits of both policy and value iteration in a single approach.

2.2.1 Value Iteration

Q-learning

Q-learning is a reinforcement learning approach that requires a Q-table, which consists of Q-values that indicate the return on an agent's action in a specific state. The Q-table has dimensions that correspond to the number of possible states against the number of possible actions in the action space. An issue that arises with this in larger environments is that the computational expense of storing, editing and accessing data in the Q-table which grows exponentially.

$$Q_{new}(s,a) = Q(s,a) + \alpha[R(s,a) + \gamma max Q'(s',a') - Q(s,a)] \tag{1}$$

After each step, the Q-values are updated as in Equation 1. These updates will be decided based on the reward associated with the outcome of the action in its previous state. Several parameters are used to determine the updated Q-values.

- The learning rate, α, is a value between 0 and 1, which controls how quickly the model adapts to the problem. A higher value will place more emphasis on the update to the Q-value. Conversely, a lower value results in slower updates to the Q-values.
- The discount factor, γ, is used to balance the current and future rewards. A higher

discount factor results in the model placing a larger emphasis on future rewards in the long term, whereas a lower discount factor results in a model that places emphasis on immediate rewards.

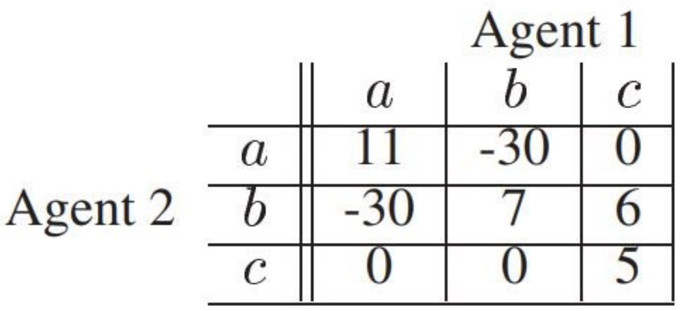
- The reward, $R(s, a)$, is a parameter set by the user, which varies greatly depending on the environment.

There are two types of Q-learning behaviour - exploitation and exploration. In exploitation behaviour, depending on the state the agent is currently in, the agent will perform the action that has the highest Q-value. While this would intuitively provide the desired behaviour of the agents, due to the nature of Q-learning, systems are typically initialised with no prior knowledge of how to behave. The optimal behaviour is learnt over time through the rewards assigned to the agent after executing actions in states.

Exploration is when the action is chosen at random. Exploration is useful in the early stages of the learning process to overcome the initial zero values. This behaviour can be controlled with an additional parameter, ϵ. ϵ is set to a percent and determines how much the agent will explore. A random number between 0 and 1 is generated, and if it is less than ϵ, then a random action is chosen. Otherwise, the action with the highest Q-value is chosen. This value is often set closer to 1 and is then decayed over time with each episode, providing the desired behaviour of exploring initially, then refining which action is the most beneficial.

Kapetanakis and Kudenko investigated a modified Q-learning approach in a cooperative multi-agent system [13]. Prior to this research conducted in 2002, reinforcement learning approaches for independent agents in a system could not guarantee convergence to the optimal solution. This flaw in performance severely limited the application of reinforcement learning in multi-agent systems due to the lack of certainty in achieving optimal performance. In Kapetanakis and Kudenko's investigation, two applications in which coordination between agents was quite challenging were explored - cooperative matrix games known as the penalty game and the climbing game, introduced in 1998 [14].

Both applications require two agents to cooperate to achieve optimal performance, based on the reward received after the joint action. Figure 3 details the reward function for the climbing game and highlights a significant issue that arises with strongly penalised actions. In a situation where one agent selects a part of the optimal joint action (a,a), while the other selects a different option, such as b, the severe penalty could prevent the agent from selecting a in future, thus being detrimental to the agents' learning process. The agents could then veer towards selecting options b and c, regardless of the smaller reward, due to the safety associated with not incurring a large penalty. This is known as the pareto-optimal solution, where the system converges to the suboptimal solution [15]. This arises in multi-objective optimisation problems, where a range of objectives are trying to be satisfied simultaneously. Pareto-optimal refers to a solution that cannot improve the performance associated with one objective without negatively impacting another objective [16].



		Agent 1		
		a	b	c
Agent 2	a	11	-30	0
	b	-30	7	6
	c	0	0	5

Figure 3 Climbing game [13]

A study by Deng et al. 2020 proposes an online task allocation algorithm that is based on Q-learning. A model-free algorithm was used to learn the strategy for task allocation in the system. A set number of tasks are allocated in each stage, and the episode is terminated once these tasks are completed. In this study, the greedy strategy is employed, with a small probability of a random action, epsilon. The use of the Q-learning algorithm in the system reduced the overall completion time and system cost. This algorithm shows definitive outperformance of other algorithms, regardless of the number of tasks.

Another application of Q-learning in a task allocation environment is from Jiang et al. [17]. In this study, objectives of minimal energy consumption and timely completion of tasks assessed. A trade-off between the two objectives determines the allocation of tasks to particular vehicles in the system. A Q-learning approach is taken whereby a two-phase solution is proposed. The first phase establishes the type of task through a trained Bayesian classifier, from which the mode of operation is determined. The second phase is where the Q-learning is implemented, which is done by assigning certain resources based on the mode operation. The results of this study show that the modifications to the task allocation system result in a reliable and significant improvement in performance, while also reducing the cost associated with the system.

From these reviews and applications of Q-learning, it is evident that there are certain shortcomings and flaws associated with traditional Q-learning as a model for reinforcement learning. While it excels in particular areas that other methods may fall short in, overall, much improvement could be made. This gives rise to more advanced and complex variations of Q-learning such as deep Q-learning, double Q-learning, delayed

Q-learning and Greedy GQ.

Hindsight Experience Relay (HER)

Hindsight Experience Replay (HER) is a reinforcement learning technique introduced by Andrychowicz et al. in 2017 [18]. The general process is that the current policy is run, and all observations are stored in the replay experience buffer. A certain number of goals are sampled based on the states which were visited in the episode. For each objective, the states, actions and rewards are also stored in the buffer with the sampled goal and the reward associated with the sampled goal. These steps are then repeated a specific number of times to populate the experience buffer. Following this, a certain number of the observations are sampled from the buffer to train the network using the experience gathered from previous episodes.

Every transition is stored in the replay buffer after each episode. The objective will influence the agents' actions but will not affect the environment; therefore, each trajectory can be replayed given that an off-policy algorithm is used.

Experiments using this type of reinforcement learning have shown that it is effective, with an ability to learn from an environment with sparse rewards. Andrychowicz et al. conducted experiments on a robot arm learning specific tasks - pushing, sliding, picking up and dropping. These resulted in an excellent performance with a high success rate, indicating that the incorporation of this into existing algorithms could significantly improve their performance [18]. However, an issue with these results is that excellent performance only occurs with carefully tuned parameters and specific reward functions. Therefore, variation in any aspect may cause the performance to deteriorate when applying this technique to different situations.

Deep Q-learning (DQN)

Due to the nature of Q-learning, which requires storing and accessing a large amount of data in more complex environments, a proposed solution to mitigate the increasing computational intensity is to employ a deep neural network to estimate the Q-values instead of storing values for each possible state-action combination. This approach is known as Deep Q Learning (DQN). It has shown proven improvement in performance in a multitude of applications, including task allocation.

A drawback particularly evident in deep reinforcement learning is the dynamic target. Compared to deep learning, in which the target is stationary, the target in deep RL changes with each iteration. This issue is mitigated through experience replay and the use of a target network.

```
1  Initialize replay memory $\mathcal{D}$ to capacity $N$
2  Initialize action-value function $Q$ with random weights
3  for episode $= 1, M$ do
4      Initialise sequence $s_1 = \{x_1\}$ and preprocessed sequenced $\phi_1 = \phi(s_1)$
5      for $t = 1, T$ do
6          With probability $\epsilon$ select a random action $a_t$
           otherwise select $a_t = \max_a Q^*(\phi(s_t), a; \theta)$
7          Execute action $a_t$ in emulator and observe reward $r_t$ and image $x_{t+1}$
8          Set $s_{t+1} = s_t, a_t, x_{t+1}$ and preprocess $\phi_{t+1} = \phi(s_{t+1})$
9          Store transition $(\phi_t, a_t, r_t, \phi_{t+1})$ in $\mathcal{D}$
10         Sample random minibatch of transitions $(\phi_j, a_j, r_j, \phi_{j+1})$ from $\mathcal{D}$
11         Set $y_j = \begin{cases} r_j & \text{for terminal } \phi_{j+1} \\ r_j + \gamma \max_{a'} Q(\phi_{j+1}, a'; \theta) & \text{for non-terminal } \phi_{j+1} \end{cases}$
12         Perform a gradient descent step on $(y_j - Q(\phi_j, a_j; \theta))^2$ according to equation 3
13     end for
14 end for
```

Figure 4 Deep Q-learning algorithm with experience replay [19]

The overall process for DQN can be seen in Figure 4 from [19]. The estimated Q-values are initialised to 0 for all state-action pairs. Using the epsilon-greedy policy, an action is selected and executed, causing the agent to move into a new state and receive a reward. This transition is then stored in the replay buffer, followed by sampling a random batch of transitions in the existing replay buffer. The loss is calculated by the equation shown on line 12 of the algorithm. Gradient descent is then performed on the network parameters to minimise this loss. This is repeated for a certain number of iterations, after which the weights in the neural network are copied to the weights of the target network. Since convergence is not guaranteed in deep reinforcement learning, this is repeated for a fixed number of episodes instead of until convergence.

Dai et al. employed a deep q-learning approach to their task allocation problem, where the agents in the system explore an environment and can be required to destroy specific targets [20]. The learning-based approach is desirable for multi-agent systems, specifically in unfamiliar, dynamic environments. In this application, the DQN did not perform well; however, combining the advantages of several approaches could be beneficial to find the trade-off between certain objectives - minimising completion time or minimising travel cost.

A study with a very relevant application of deep reinforcement learning is Malus et al. [21]. A DQN is used to determine the bids placed by agents in an environment where transportation tasks are to be completed. The approach is distributed and focuses on the dispatch of tasks, not navigation or routing. The results of this were promising, showing that the policy was successfully learned over the simulation. A shortcoming of this study was the lack of experimentation on a practical system, as all the results were simulated.

While the performance of the DQN in the first application shows significant room for improvement, the second application shows much promise. The non-stationary target allows for the potential situation where the system cannot converge due to instability, which is a significant drawback compared to other approaches where convergence is guaranteed [21]. However, the ability to modify the algorithm in certain aspects could allow for major improvements in the application of DQN in task allocation environments.

2.2.2 Policy Iteration

A policy refers to the action that the system is recommended to take in any given state. This is typically assigned the symbol π. Policy iteration refers to converging to the optimal policy, ensuring that the action to be taken is the most beneficial to the entire system.

Policy Gradient

A popular policy iteration algorithm is known as Policy Gradient, whereby a specialised policy score function measures the quality of a policy. The concept of 'policy gradient ascent' is used to determine the best parameter that improves the policy. The general process is as shown in Algorithm 1.

Algorithm 1: Policy gradient algorithm

```
while objective not reached do
    observe current state of the environment
    execute action based on the policy
    enter into new state
    take further action based on the observed state
    adjust the policy based on total rewards received
end
```

The expected rewards are calculated by the sum of the probability of a trajectory multiplied by the corresponding rewards. The objective of the algorithm is to establish a policy that would create a trajectory that maximises the aforementioned expected rewards. This is an online method, which means that the training data must be obtained through the current policy.

Pippin and Christensen propose a reinforcement learning method to detect which agents are more likely to submit a bid that accurately reflects the cost of performing a task [22]. In this study, the policy is initialised. An algorithm is used to learn a single parameter set to 1, reflecting the belief that each agent will perfectly estimate the tasks it performs. A random set of permutations is generated for the policy by either adding or subtracting epsilon in the main loop. Once all of the permutations have been evaluated by the system, the gradient is approximated by calculating adjustments related to each parameter in the policy. For each parameter, the step and global step distances are applied to the adjustment to normalise the values. The policy is then updated, and adjustments to the cost factor are calculated by the algorithm using the reward function. Experiments from the study show that the learning mechanism can be particularly effective in detecting team members with poor performance in the auctions. Furthermore, the results prove an increase in efficiency of task allocation through learning the performance characteristics of individual agents in the system. However, the learning mechanism is limited and so additional relevant mechanisms should be considered for further improvement in task allocation performance.

Advantage Actor-critic (A2C) and Asynchronous Advantage Actor-critic (A3C)

The advantage actor-critic algorithm is commonly used in reinforcement learning approaches. The process can be visualised in Figure 5, provided by Steinbach [23]. The network is initialised with random weights, and the current policy is used, where each of the state-action transitions are saved. The reward, policy loss and value loss are calculated and further used in stochastic gradient descent over each batch. The advantage is calculated by subtracting the expected value of the state from the Q-value of that state. If the return is better than expected, the probability is increased, and similarly, if the return is worse than the expected value, the probability is decreased.

The system repeats selecting actions based on the current policy and calculating certain values until the policy has converged, resulting in the system's optimal performance.

The asynchronous advantage actor-critic algorithm is similar to the A2C algorithm, the main difference being that the A3C algorithm has independent agents, with separate networks, all interacting in parallel. This allows for more training data to be collected due to the significantly larger portion of the state-action space that can be explored compared to running a single network.

Feng et al. developed a cooperative computation offloading and resource allocation framework [24]. The proposed framework includes a multi-objective function to maximise the computation rate of mobile edge computing systems. This A3C-based algorithm handles the complexity and dynamics of the environment. Significant performance improvement was achieved by the algorithm, with prompt convergence.

```
Algorithm TD Advantage Actor-Critic
Randomly initialize critic network $V_\pi^U(s)$ and actor network $\pi^\theta(s)$ with weights $U$ and $\theta$
Initialize environment E
for episode = 1, M do
    Receive initial observation state $s_0$ from E
    for t=0, T do
        Sample action $a_t \sim \pi(a|\mu,\sigma) = \mathcal{N}(a|\mu,\sigma)$ according to current policy
        Execute action $a_t$ and observe reward $r$ and next state $s_{t+1}$ from E
        Set TD target $y_t = r + \gamma \cdot V_\pi^U(s_{t+1})$
        Update critic by minimizing loss: $\delta_t = \left(y_t - V_\pi^U(s_t)\right)^2$
        Update actor policy by minimizing loss:
            $Loss = -log\left(\mathcal{N}(a \mid \mu(s_t), \sigma(s_t))\right) \cdot \delta_t$
        Update $s_t \leftarrow s_{t+1}$
    end for
end for
```

Figure 5 Advantage actor-critic algorithm [23]

While A3C has shown excellent performance, it is particularly useful in complex state and action spaces, which introduces a level of complexity that is not particularly necessary in task allocation problems.

2.3 Multi-agent Reinforcement Learning

Advancements in reinforcement learning have unearthed the necessity for a more complex solution to certain applications. For instance, certain gameplays require more than a single agent to participate. Other applications in cyber-physical systems also tend to require a solution that is modelled with multiple agents interacting with the environment and each other.

The algorithms used in multi-agent reinforcement learning (MARL) are typically categorised into two groups; cooperative and competitive. However, some studies have shown that a combination of the two can be beneficial in certain applications.

[25] demonstrates an application of MARL for energy management. A MARL approach is used to schedule the operation of various components to meet the demands in a residential energy system. The energy management is modelled as a non-linear optimisation problem, which employs reinforcement learning to solve it. Since it is a multi-agent system, each agent constructs an individual Q-function for learning and development. The results of this study showed a lower cost policy, suggesting improvement in the performance. The approach could adopt non-cooperative policies to facilitate the competitive allocation of resources between various systems to improve this further.

A challenge that arises with multi-agent systems, which is not encountered with single-agent systems, is multiple objectives for each of the different agents. With each of the different agents and their respective objectives, whether the system is cooperative or competitive, the environment becomes non-stationary. Each agent updates the environment which it observes in parallel, and due to this, the optimal solution can often be missed [25].

Another application of MARL is in [26], where the approach is a fully decentralised MARL with networked agents. The agents share a collective goal of maximising the globally averaged return through agent-to-agent communication. The framework proposed in [26] modifies and extends from existing approaches by modelling the networked agents with local sensing capabilities to realise the decentralised approach. The simulation results exhibit successful convergence with non-linear function approximators, such as a neural network for approximation.

[21] demonstrates a similar scenario with MARL used to solve a task allocation problem. While it proved successful for the simulation and experimentation of the provided scenarios, it was not suitable for more complex environments.

3. Problem and Methods

3.1 Problem Definition

The proposed approach uses multi-agent reinforcement learning (MARL) to allow for adaptability of the system to a dynamic environment, solving issues that can arise due to the layout or obstacles along the AMR's route. Each AMR is represented by an agent that learns and bids on the available task based on their current observation of the environment.

A	set of all agents
T	set of all tasks
n	total number of agents
m	total number of tasks
j	index of jth agent
i	index of ith task
A_j	index of jth agent
T_i	task at index i
k	number of tasks to look ahead
t_k	set of tasks in "look-ahead"

The following assumptions are made in the problem:

1. All agents are available from initialisation (time is 0)
2. The system initialises with tasks in the task queue
3. Each agent can only process one transportation order at a time
4. The agents have built-in vision and obstacle avoidance

The task allocation problem is formulated as such: Let A = $\{A_1, ..., A_n\}$ be the set of agents, and T = $\{T_1, ...T_m\}$ be the set of tasks. The goal of the task allocation problem is to determine the optimal assignment of tasks to the available agents depending on their locations and current tasks assigned. Once all m tasks have been assigned and completed, the episode is considered to be finished, and the process repeats.

A task is described by the tuple shown in Equation 2, where x_s and y_s represent the start x and y coordinates of the task, x_e and y_e represent the end x and y coordinates, d_T represents the deadline expressed in absolute iterations, $create_T$, $start_T$ and $complete_T$ represents when the task was created, started and completed respectively, $ability_T$ represents the ability level required for the task, and id_T is the task ID. Each agent has an ability level of [0,1,2], which can be translated to weight carrying capabilities in a real-life scenario.

$$T = \{x_s, y_s, x_e, y_e, d_T, create_T, start_T, complete_T, ability_T, id_T\} \quad (2)$$

A task is a transportation order, whereby the agent must go to the start position and transport the materials to the end position. Once the agent has reached the end position, it is considered to be completed.

The environment is modelled as a grid of size 10 x 10. The agents can move in 4 directions in the simulation as the added complexity of combined actions was not deemed necessary for this research.

The main metrics used to quantify the performance of the model are the delay and the distance travelled. The delay is defined as the time between creating the task and the completion of the task. The mathematical representation can be found in Equation 3. This is an important performance metric as in real-world applications of this model, certain tasks would be time-sensitive and require completion by a specific deadline. Furthermore, completing tasks as early as possible is generally desirable in most contexts. The distance travelled is normalised and defined as the total distance travelled by the agent divided by the distance between the start and endpoints of each of the tasks completed by the agent. This can be seen in Equation 5. Minimising this value is crucial as it promotes an efficient system by reducing and eliminating unnecessary travel. Another metric used was the lead time, as seen in Equation 4, which indicates the time between the iteration on which the task is created and the iteration on which it is completed. This gives insight into the overall time taken for the agent to complete the tasks it is assigned and to minimise this across all agents.

$$DELAY = \sum_{o}^{O'} (time_{completed} - time_{deadline}) \tag{3}$$

$$LEADTIME = \sum_{o}^{O'} (time_{completed} - time_{created}) \tag{4}$$

$$NORMALISED\_DISTANCE = \frac{\sum_{a}^{A} distance\_travelled}{\sum_{o}^{O'} distance} \tag{5}$$

where:

$\sum_{o}^{O'}$ represents all completed tasks in set O'

$\sum_{a}^{A}$ represents all agents in set A

$time_{completed}$ represents the iteration on which the agent reaches the end position, completing the task

$time_{deadline}$ represents the iteration on which the task is required to be completed

$time_{created}$ represents the iteration on which the task is generated and put into the queue

$distance\_travelled$ represents the total distance travelled by the agent in that episode

$distance$ represents the distance between the start and end position of the task

The most significant issue that arises with achieving optimality in this situation is the dynamic environment and the inability to determine what the future tasks could be and how to adapt the agents' behaviour to accommodate for this.

The most significant issues that arise with achieving optimality in this situation are the dynamic environment, the inability to determine what the future tasks could be, and how to adapt the agents' behaviour to accommodate for this uncertainty. This increases the complexity associated with convergence to optimal performance of the system.

3.1.1 Proposed Model

Model Framework

This model utilises a consensus-based auction algorithm to determine the task allocation for each episode. The possible bids, B, that can be placed are between 0 and 1 at intervals of 0.1. Each agent places a bid on the task available - the bid is determined by the neural network, which approximates a Q-table. The bid value is that which corresponds to the highest Q-value. A high Q-value translates to an action with a very desirable outcome depending on the criteria set by the user.

Once a task has been assigned to an agent, it travels from its current position to the start position. A negative reward is associated with each move to simulate the desired efficiency in reaching the target in minimal steps. Once it has reached the start position, it travels to the end position, accumulating the negative reward until it reaches the end position. It receives a very positive reward, and the task is considered complete. To promote collaborative behaviour, all the agents receive a positive reward when one agent completes a task.

Deep Q Neural Network

Our proposed algorithm uses a Deep Q Neural Network (DQN), a type of model-free reinforcement learning and uses value iteration to improve the system's performance. A sequential model with two layers is used. The *Adam* optimiser from Keras is a "stochastic gradient descent method that is based on adaptive estimation of first-order and second-order moments". A linear activation function was used in lieu of a step activation function as it allowed for a more versatile output.

In this system, two models are used to improve the neural network's performance - the main model and the target model. The target model is updated every n episodes, where n is an adjustable parameter, set to 5 in this system. This means that every five episodes, the target model is reset to another model, resulting in less severe fluctuations and more stable training. The replay memory simplifies the training process, as it is quite computationally intensive to train the model every step. The past 1000 actions are stored in the "memory", and the main model is then fitted on a random selection of these actions, and eventually, the models converge.

Adam Optimiser

The Adam (Adaptive Moment Estimation) optimiser has 4 parameters that can be adjusted for use; learning rate, β_1, β_2 and ϵ. The default values of which are 0.001, 0.9, 0.999 and 1×10^{-7} respectively. These default values have proven good performance of the optimiser, and so these were used for the model. At each time step t , an exponentially decaying average of past squared gradients, v_t, is stored as well as an exponentially decaying average of past gradients, m_t, similar to momentum. These are calculated as seen in Equations 6 and 7.

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1) g_t \tag{6}$$

$$v_t = \beta_2 v_{t-1} + (1 - \beta_2) g_t^2 \tag{7}$$

m_t is the estimate of the first moment (the mean) of the gradient, and v_t is the estimate of the second moment (the uncentered variance) of the gradient. Since these are initialised as vectors of zeros, it is observed that they are biased towards zero. This is remedied by calculating the bias-corrected moment estimates, as seen in Equations 8 and 9.

$$\hat{m_t} = \frac{m_t}{1 - \beta_1^t} \tag{8}$$

$$\hat{v_t} = \frac{v_t}{1 - \beta_2^t} \tag{9}$$

These corrected estimates are used to update the parameters using the Adam update rule, as seen in Equation 10, where θ_t is the update for the parameters.

$$\theta_{t+1} = \theta_t - \frac{\nu}{\sqrt{\hat{v_t}} + \epsilon} \hat{m_t} \tag{10}$$

The parameterised value function, $Q(s, a; \theta_i)$, uses the weights of the Q-network to calculate the loss for each iteration, i, as seen in Equation 11. This loss is minimised to optimise the system, and the Q-network parameters are updated every n episodes.

$$L_i(\theta_i) = \mathbb{E}_{(s,a,e,s')}[(r + \gamma max Q(s', a'; \theta_i^-) - Q(s, a; \theta_i))^2] \tag{11}$$

Reward Function

A crucial part of Q-learning is the reward function, which essentially governs the behaviour of the entire system. In multi-objective optimisation problems, the reward function must be designed in such a way that allows each of the objectives to be achieved while maintaining a balance.

The reward function for this model was tested and modified upon several iterations for the final outcome and is available in Equation 12.

$$R(s,a) \mathrel{+}= \begin{cases} 100 & agent_{task_complete} = True \\ -60 & agent_{task_late} = True \\ -1 & agent_{move} = True \\ 50 & agent_{task_complete} = True, foranyagentinteam \end{cases} \tag{12}$$

where:

x_t represents the observation at the specified time step

EMA_t represents the exponential moving average calculated at the specified time step

α represents the smoothing factor and determines how significant the most recent data point is, this value is set between 0 and 1

TensorFlow and Keras

TensorFlow used in conjunction with the Keras library provided a means of implementing the reinforcement learning model without developing a program from scratch. A significant application in which Keras facilitates the development of the model is in the neural network. The Sequential() model from Keras is used, which constructs a plain stack of layers for the neural network. The Sequential model was favoured over the Functional API as it introduced an unnecessary complexity to the system. However, the system would remain functional if the model was changed to the Functional API, allowing for a wider range of functionality.

The Keras library provides a wide range of functions for the layers of the neural network, such as Dense(), Dropout(), Conv2D(), MaxPooling2D(), Activation(), Flatten(). The process of creating a convolutional neural network using these functions can be seen in Algorithm 2.

The Dense() function generates a densely connected layer of the neural network used in the output layer of our proposed model. The Dropout() function randomly assigns

Algorithm 2: Creating convolutional neural network using Keras

```
initialise stack of layers - **Sequential()**
for each layer do
    create convolutional layer - **Conv2D()**
    apply activation function layer output - **Activation()**
    downsample input - **Flatten()**
    dropout to prevent overfitting - **Dropout()**
end
flatten to 1-dimensional shape - **Flatten()**
create dense layer for output - **Dense()**
compile model
```

inputs to 0 at a user-specified rate to prevent overfitting. To maintain the sum over all of the inputs, the value of inputs not set to 0 is scaled accordingly. The Conv2D() function generates a convolutional layer for the neural network used for both layers of the neural network. This type of neural network was chosen due to the reliability of convolutional neural networks in history and its ready availability from the Keras library. The MaxPooling2D() function downsamples the input through a specific calculation found in the Keras library documentation. Max pooling is essential in our model. It reduces the overall computational cost associated with the system by reducing parameters; it also assists in the prevention of over-fitting. The Activation() function is necessary as it applied the specified activation function to the output. In this model, a linear activation function is used, as mentioned in Section DQN. The Flatten() function is required to process the input into the desired shape for the output, typically flattening the shape from a 2-dimensional input into a 1-dimensional output.

All these functions are used together to construct the neural network, which is done without writing the code from scratch to build the neural network.

Look-ahead Parameter

After reviewing many task allocation solutions with similar principles to the proposed model, a gap was identified in the research. While the current task available to be bid on may only be assigned to an available agent, this allowed for an inefficient allocation of tasks to agents. This would significantly affect the model's performance as the allocation scheme would assign tasks to agents with low bids and, therefore, low efficiency.

Parallel auction had been introduced, however, the computational cost of this would be high in an environment where the number of tasks was also high. Hence in an environment with 1000 tasks, the system would have to process all 1000 tasks and the reassignment of tasks to over-booked robots, resulting in significant delay and waste of computational resources.

A parameter termed the "look-ahead" parameter, k, was introduced into the system to mitigate the effects of the aforementioned downfall of existing models. The k value determines the number of tasks the system assesses when receiving bids. The system with a k value of 1 would behave in the same manner as that of a system without the addition. Any k value greater than one would invoke the look-ahead behaviour. This allowed the benefits associated with the parallel auction while mitigating the issues that would arise from a scaled system.

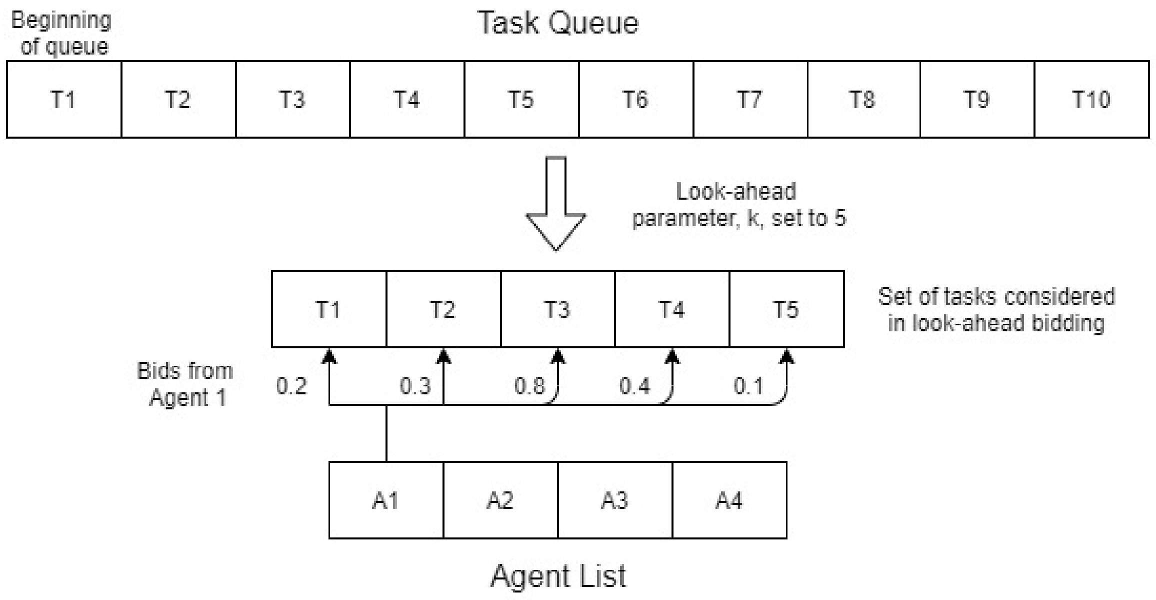


Figure 6 Visualisation of look-ahead functionality for Agent 1

To realise the look-ahead functionality, the system receives bids from agents for ’k’ tasks in the task queue, Figure 6 visualises this for Agent 1 in a system with four agents. For instance, in a system with a k value of 5, each agent produces a bid for the first five tasks in the queue. Each of these bids is assessed, and only the highest bid is sent to the allocation system. Any bid produced lower than the highest bid is sent as a 0 to remove the possibility of that task being assigned to the agent. Tasks are then assigned as per the original scheme, where the highest bidder receives the task.

Incorporating this functionality works to prevent a first-in-first-out system that relies heavily on availability and does not properly assess the agent’s current position, situation and environment. An instance in which this would significantly impact performance is where only one agent is available, and the rest are occupied, and the task is not well-suited for the available agent. For the system without look-ahead functionality, any non-zero bid would be sufficient to assign the task to that agent. However, this means that an opportunity for a higher bid from that agent on one of the upcoming tasks would be missed. Conversely, a system with a look-ahead value of 10 would ensure that only the most suitable task would be assigned to the available agent of the ten upcoming tasks in the queue. For instance, if the available agent were to bid 0.1 on the first task and 0.8 on a task later in the queue, only the 0.8 bid would be considered by the system, thus preventing inefficient allocation of resources.

An example of the look-ahead functionality can be observed in Table 1 and Table 2.

In the example, T_1 is assigned to A_3, T_2 is assigned to A_4. Since the bids for T_3,T_4 and T_5 are all 0, they remain in the queue for the next round of bids, as seen in Table 3.

In this case, without the "look-ahead" method, T_3 would have been assigned to A_2 as it is the only non-zero bid at the time. However, the low bid of 0.1 indicates that it is far from the best candidate to complete the task. Further, without the "look-ahead" method, T_4 would have been assigned to the highest bidder A_2 for a relatively low bid of 0.3. With the "look-ahead" method in place, the bids in 4 show that T_3 is assigned to A_1 for a high bid of 0.8 and T_4 is assigned to A_3 for another high bid of 0.9. Hence, the use and implementation of this method is beneficial to the task-allocation system.

It can be easily observed that the lower bids that would have been accepted for assignment are nullified, thereby facilitating agents to be assigned to more suitable tasks.

Table 1 Bids for tasks in "Look-ahead" set of tasks - Iteration 1

		Tasks				
		T1	**T2**	**T3**	**T4**	**T5**
Agents	**A1**	0	0	0	0	0
	A2	0.1	0.6	0.1	0.3	0.2
	A3	0.7	0	0	0.1	0.3
	A4	0.4	0.8	0	0.2	0

Table 2 Compiled agents bids after processing (left), compiled bids for each task (right) - Iteration 1

A1	(0, 0, 0, 0, 0)	**T1**	(0, 0, 0.7, 0)
A2	(0, 0.6, 0, 0, 0)	**T2**	(0, 0.6, 0, 0.8)
A3	(0.7, 0, 0, 0, 0)	**T3**	(0, 0, 0, 0)
A4	(0, 0.8, 0, 0, 0)	**T4**	(0, 0, 0, 0)
		T5	(0, 0, 0, 0)

Table 3 Bids for tasks in "Look-ahead" set of tasks - Iteration 2

		Tasks				
		T3	**T4**	**T5**	**T6**	**T7**
Agents	**A1**	0	0	0	0	0
	A2	0.1	0.6	0.1	0.3	0.7
	A3	0.7	0	0	0.1	0.3
	A4	0.4	0.8	0	0.2	0

Table 4 Compiled agents bids after processing (left), compiled bids for each task (right) - Iteration 2

A1	(0, 0, 0, 0, 0)	**T3**	(0, 0, 0.7, 0)
A2	(0, 0, 0, 0, 0.7)	**T4**	(0, 0, 0, 0.8)
A3	(0.7, 0, 0, 0, 0)	**T5**	(0, 0, 0, 0)
A4	(0, 0.8, 0, 0, 0)	**T6**	(0, 0, 0, 0)
		T7	(0, 0.7. 0, 0)

Individual Queues

Another concept that was not thoroughly explored in literature was the use of queues in a task allocation system. Certain models allowed one task to be held in the agent's task queue [21], while others did not allow for this, rather just the task currently being executed. However, the impact of this queue on performance had not been assessed.

Two types of queues were assessed for use in the proposed model. The first type was dynamic while allowing for assignment of tasks in an agent's individual queue, also allowed for reassignment of those tasks to other agents in the case of a higher bid. While this would ensure that tasks are not held stagnant in a queue while a better agent is available, the high computational intensity did not compensate for the impact on performance. Furthermore, the complexity of implementing this type of queue was unnecessary due to the similarity in the behaviour of the system to a system that did not allow for tasks to be assigned to the agents' individual queues.

The second type of queue was static, where once the task was assigned to the agent's queue, it could no longer be reassigned within that session. A drawback of this type is that the tasks are able to sit in the queue unexecuted for extended periods, depending on the length of the queue. However, the model's design is such that the bid depends on the end position of the task that is to be completed directly prior to the task currently up for bid. Therefore, the overall execution would result in the most efficient assignment of tasks, minimising the distance required to travel between task completion and task execution.

Based on the assessment of the two types, the static type was implemented as it did not significantly increase the complexity of the model while allowing for an efficient means of assigning tasks.

Multi-agent Reinforcement Learning System

To ensure the collaborative behaviour of the model and realise a multi-agent learning model, the agents must understand that any completion of a task has a significant positive reward for the entire system. Suppose each agent were to focus solely on their own reward and only improve in individual performance. In that case, this could result in a competitive nature within the team, where agents are greedy and behave selfishly. While this has been successful in the past, certain scenarios would benefit from a holistically cooperative team, with the proposed scenario as an example.

To realise this in the proposed model, the reward function was developed in such a way to positively impact all agents when one agent completed a task. While the individual agent will learn and train based on the quality of the execution, the rest of the agents will only be positively rewarded. This functionality is set up to promote cooperation amongst the robot team and showed positive results in [21].

3.2 Experimental Setup

3.2.1 Simulation Framework

The simulation was set up with an episodic horizon, with the environment resetting with each episode, reinitialising the agents within the system upon completion of the tasks required. This was intentionally done to accommodate the output of the metrics and performance of the system over time. Using episodes in lieu of a single horizon was beneficial in tracking the improvement or degradation of the system with each episode. This allowed for earlier insight into the effects of any modifications to the model, reducing wasted computation time on additions detrimental to the performance. Algorithm 3 provides an overview of the simulation.

Order dispatching is the main focus of this research. Hence, the execution of the assigned tasks (agents travelling from the start to the end positions) is primitive. It incrementally matches the x-axis of the agent and the destination and then the y-axis, as seen in Figure 7. The blue arrow indicates motion in the x-axis, while the orange arrow indicates the agent travelling in the y-axis. In the application for which this algorithm is intended, the robots will have vision while travelling. Therefore, obstacle avoidance and collision prevention are not incorporated into the functionality of this algorithm.

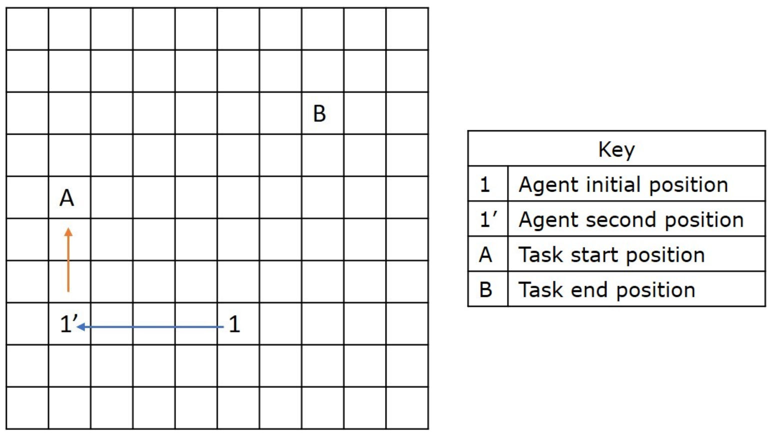


Figure 7 Motion of agent during task execution

Each of the agents behaves independently; however, the knowledge from all the agents is shared. The neural network is trained with each agent, with each bid placed. This speeds up the learning process and allows the agents to inherently work collaboratively with each other.

Another essential feature of the simulation is the repeatability of the results. This repeatability and consistency for performance comparisons across the varying parameters

Algorithm 3: Simulation framework

```
for each episode do
    initialise variables and task queue
    while tasks incomplete do
        generate task
        reset lists (bids, actions etc.)
        if new task generated then
            add task to queue
        end
        if tasks in queue then
            for each task in queue in range k do
                for each agent do
                    if random > epsilon then
                        choose bid from highest Q-value
                    else
                        choose random bid
                    end
                    compile bids
                end
            end
            for each agent do
                set all bids below highest to zero
            end
            for each task in queue in range k do
                assign task to highest bidder
                if all bids zero then
                    return task to queue
                else
                    remove task from queue
                end
            end
        end
        for each agent do
            compute adjust for parameters
            adjust weights in neural network
        end
    end
    log values and tasks
    decay epsilon
end
save model to file
```

was achieved through the tasks available in the queue. For each simulation, the same tasks were generated, and so the performance would vary based on the randomly assigned initial positions, as well as the difference in parameters - number of agents, number of tasks, look-ahead and individual queue parameters. A seed was crucial in realising this as the task parameters were randomly generated.

Table 5 Agent-task parameter pairs

		Tasks, T	
		20	10000
Agents, A	4	4A20T	4A1000T
	10	10A20T	10A1000T
	20	20A20T	20A1000T

Table 6 Individual queue and look-ahead parameter pairs

		Individual queue length, N				
		N0	N1	N2	N3	N4
Look-ahead parameter, K	K1	K1N0	K1N1	K1N2	K1N3	K1N4
	K5	K5N0	K5N1	K5N2	K5N3	K5N4
	K10	K10N0	K10N1	K10N2	K10N3	K10N4
	K20	K20N0	K20N1	K20N2	K20N3	K20N4

The experiments were set up as follows, with a summary that can be found in Table 5 and Table 6. For each pair of varying agents and varying tasks per episode, the corresponding pairs from the look-ahead and individual queue parameters were tested. Referring to the tables, each cell in Table 5 was paired with all of the cells in Table 6.

The number of agents was varied with the following parameters, [4, 10, 20], which was paired with the number of tasks per episode [20, 1000]. This was done to ensure that the model was viable for larger environments for scalability and robustness.

The look-ahead parameter was set with the values [1,5,10,20], while the values for the individual queues for the agents was set to [0,1,2,3,4]. Varying these values was necessary to ensure that the improvements from the look-ahead feature were observable. Furthermore, the advantage of the ability to dynamically reallocate tasks is observable with the varying individual queues.

The different experiments will be referred to in the following format for the remainder of the report.

For a system with 4 agents, 20 tasks, the test will be referred to as 4A20T. For a system with a look-ahead parameter of 5, K5 will be appended to the system name as such, 4A20T_K5. Similarly, if the system has a specific individual queue length parameter of 3. this would then be appended to the system name, resulting in 4A20T_N3

3.2.2 Simulation Specifications

The model was constructed using Python 3.7 in conjunction with Keras 2.3 and TensorFlow version 1.15. The latest version of TensorFlow was not used as this version allowed for ease in switching functionality between the GPU and CPU capabilities. Furthermore, future versions of TensorFlow were not compatible with some devices, as a graphics card was required, which was not readily available in all the devices used. This was particularly useful due to the varying devices used for experimentation.

The simulations were run on a variety of computers with varying hardware and software specifications. The details of each of the computers used can be found in Table 7.

Table 7 Specifications of computers used for experimentation

	OS	**CPU**	**GPU**	**RAM**
Desk computer	Windows	i7	N/A	32GB
Supercomputer	Linux	i9	RTX3090	128GB
Virtual machines	Linux	16 VCPUs	Tesla T4	48GB

The different experiments had varying numbers of episodes dependent on the number of tasks required to complete the episode. Specifically, since there were two options for the task parameter [20,1000], the number of episodes would be 1000 for the sessions with 20 tasks and 50 episodes for the sessions with 1000 tasks.

This was done since the simulations with a smaller number of tasks required a more extended period to train the agents. Similarly, the simulations with a larger of tasks per episode experienced extended training in a shorter number of episodes, thus requiring a significantly reduced number.

3.2.3 AMR System

The model was also tested in a physical experiment developed by O Brien and Wagstaff [27], where the bidding model was integrated into their AMR system for testing and experimentation. The design of the AMRs and the software architecture is described below.

AMR Specifications

The system was designed with the ability to accommodate a range of heterogeneous AMRs in a fleet. The TurtleBot3 Waffle Pi was a suitable choice for experimentation as it met the requirements for the system to run, with vision and computation capabilities integrated into the device, and did not require additional hardware components for the intended functionality.

The AMR available for experimentation was the TurtleBot3 Waffle Pi robot equipped with a 360° LiDAR sensor for SLAM and navigation. The on-board computer was a Raspberry Pi 3, used in conjunction with an OpenCR controller, a 32bit ARM Cortex M7. Further specifications of the hardware include a Raspberry Pi Camera, servo motors for wheel control, a Li-Po battery rated at 1800mAh, and a Bluetooth module available for remote control of the device.

The robot has a maximum translational velocity of 0.26m/s, with a maximum weight capacity of 30kg. While the robot takes 2.5 hours to charge, the robot can run on a full charge for 2 hours before the battery is drained. One of the robots was used with the provided LiPo battery; however, a replacement battery was used due to some hardware complications. This had a higher rating of 8400mAh, which provided a run time over four times longer than the supplied battery.

Software Architecture

Each of the AMRs in the system have individual responsibilities, including path planning, inter-device communication, task bidding and task allocation.

The software on the AMRs consist of the following components: Robot Operating System (ROS), AMR Controller, Receiver, Bid Generator, AMR Driver and the SLAM Node. The AMRs utilise the open-source framework, ROS, to connect the system nodes, gather information from the motors and other sensors available on the device. The AMR Controller is the core node that facilitates communication between the AMR node and other devices. The AMR Receiver receives bids and only allows the AMR to bid for a single task at any given time. However, once the bid received is determined to be lower than the highest bid, it begins bidding on the next task. The Bid Generator calculates the bid that the AMR sends to the controller that allocates the task. This bid is generated using the proposed model, utilising the DQN to produce the bid based on its status in the current environment. The AMR Driver receives details of an assigned task from the AMR Controller and produces the required information for the movement. The SLAM node receives this information and constantly updates the travel path of the AMR to reach its desired destination. This is an essential node as it controls the detection of objects by the device sensors, allowing for obstacle avoidance in its path planning and movement.

Server and Database

The server also consists of several nodes, such as the AMR Handler, Task Handler, and Broker Nodes. The AMR Handler manages the communication of data that concerns the devices, and each device updates its position periodically to the database. The Task Handler controls the task assignment to the AMRs and receives any updates to the task information, referring to changes to the task's attributes. The Broker nodes operate using the Zero Messaging Queue (ZMQ) protocol to generate and control sockets. The Task, Bidding, and Command Broker nodes communicate and receive specific data from the AMR network.

The database used to store information regarding the tasks was realised through the use of MySQL. Several tables in the database include AMR, Task, Basket and Part, each containing information specific to the respective titles.

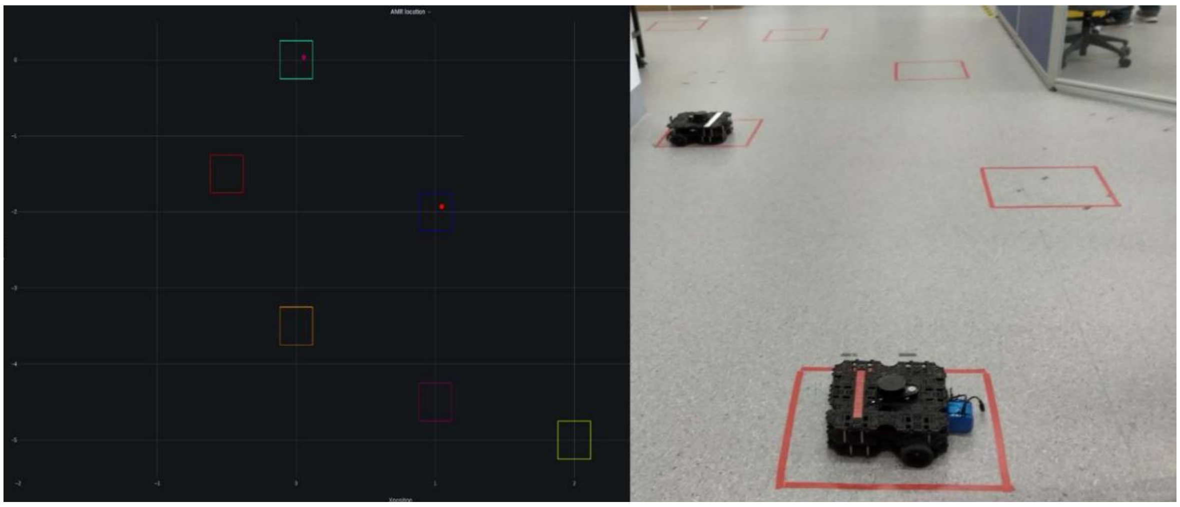


Figure 8 Testing environment for AMRs and display on dashboard [27]

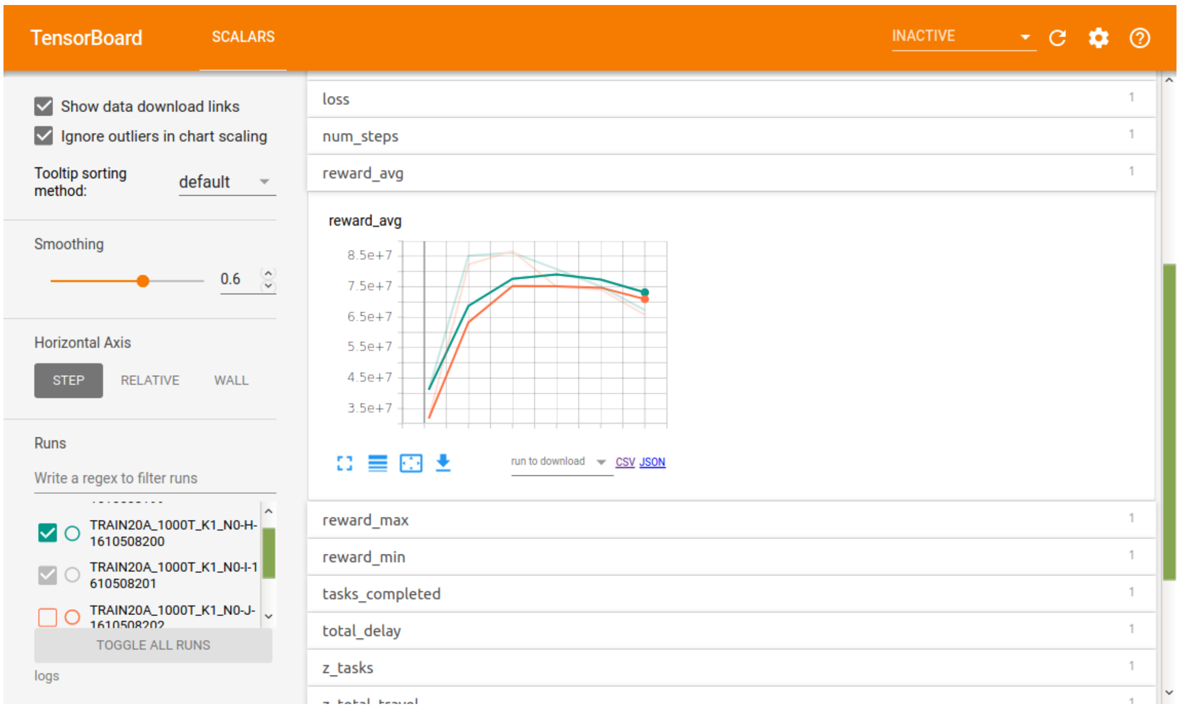
Experimentation and Results

To evaluate the performance and operation of the system, the testing environment was set up as follows. Masking tape was used to mark areas on the ground to identify specific areas that the TurtleBots would travel between, as seen in Figure 8. Each marked area referred to a specific coordinate location, to which the task would require the agent to travel. At these locations, the AMRs would communicate with the specified external device to simulate the task being performed. Considering there were a limited number of devices available for testing, the locations in the testing scenario referred to the same computer. While there were some shortcomings in the vision and obstacle avoidance area, these can be attributed to the AMRs and not the task allocation model. The conclusion drawn from this experimentation regarding the task allocation model is that it performed well under tests for robustness and scalability, where the performance was not inhibited by the addition of agents at a scale of hundreds.

4. Results and Discussion

4.1 TensorBoard

The built-in dashboard for TensorFlow, TensorBoard, was used to display the metrics during experimentation. The TensorBoard was slightly modified to fit the requirements of the model and allowed the real-time, customisable output of the results, which is pictured in Figure 9.



Tooltip sorting method: default

Smoothing 0.6

TRAIN20A_1000T_K1_N0-H-1610508200
TRAIN20A_1000T_K1_N0-I-1610508201
TRAIN20A_1000T_K1_N0-J-1610508202

Figure 9 TensorBoard dashboard used to display experimental results

In order to analyse the data output from the tests and extract meaningful results, the trend of the data is a better indicator of performance compared to the visible variation. The smoothing function used in the TensorBoard interface utilises an exponential moving average (EMA).

$$EMA_t = \begin{cases} x_0 & t = 0 \\ \alpha x_t + (1-\alpha)EMA_{t-1} & t > 0 \end{cases} \tag{13}$$

where:

x_t represents the observation at the specified time step

EMA_t represents the exponential moving average calculated at the specified time step

α represents the smoothing factor and determines how significant the most recent data point is, this value is set between 0 and 1

The EMA method causes older data to hold less weight over time, which can be observed in Equation 4.1. This is particularly useful in this application, given the nature of the simulation. Since the exploration-exploitation parameter is decreased as the episodes progress, the metrics associated with the more recent episodes hold more relevance to the trend. Furthermore, this method neglects random fluctuations and variations in the data to showcase the underlying behaviour.

The smoothing function used on the data was implemented in MATLAB, and the effect of it with different smoothing factors can be observed in Figure 10. It is evident that the smoothing factors closer to a value of 1 appear to have a near negligible effect on the data, where values closer to 0 smooth the data significantly.

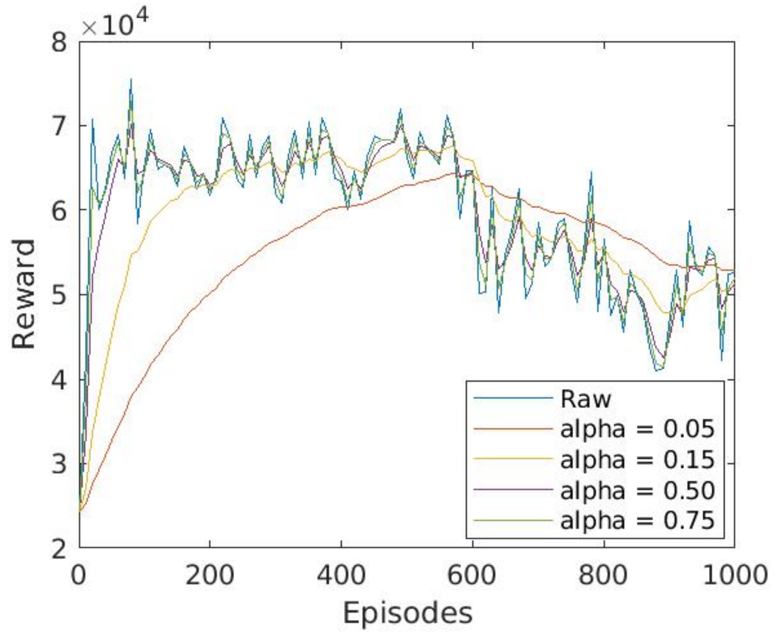


Figure 10 Data smoothed with different smoothing factors, α

4.2 Task Log

Each experiment that was run began with the tasks seen in Table 8. The results of the 4A20T_N0 and 4A20T_K5 are used as extracts for this analysis to demonstrate noticeable and meaningful trends in the data.

In the first episode of each experiment, the first task in the queue has the start position of (2,9), an end position of (1,4), a deadline at iteration 38 and a required ability of 0 in the ability scale of [0,1,2]. To observe the outcome of each task allocation process, data regarding the start time, end time, and the agent's position at the time of assignment was output into log files. For the 4A20T experiment, the agent to which the tasks are assigned can be seen in Table 9. In episode 1, the neural network is not trained, so the optimal assignment is unknown. The actions this early in the experiment are heavily influenced by randomness due to the prominence of the exploration behaviour, as well as the neural network having been initialised with random values.

Table 8 Attributes of initial three tasks for each simulation

Episode	Start Pos	End Pos	Creation time	Deadline	Ability
0	2,9	1,4	0	38	1
0	7,7	6,3	0	38	1
0	0,6	6,9	0	35	2

In episode 1, it can be observed that the task with a start position of (2,9) is assigned to an agent with a current position of (0,4), which with the designed navigation process requires the agent to travel a total of distance of 7. Compared to a task in episodes 170 and 868, where the task start position is also (2,9), the assigned agent has less distance to travel to reach (2,9). The current positions in episode 170 and 868 are 2 and 3, respectively. This shows a tremendous reduction in the distance travelled, of over 70% compared to initial performance, over a short period of 170 episodes.

Table 9 Assignment data of initial three tasks

Start time	Completion time	Agent ID	Agent Position at assignment
1	15	2	0,4
2	18	4	0,4
15	31	1	2,2

Table 10 Assignment data of tasks with identical start positions to initial tasks

Episode	Start Pos	End Pos	Creation time	Deadline	Ability	Agent ID	Agent Position at assignment
170	2,9	9,7	5	164	0	3	1,8
868	2,9	5,6	2	42	2	4	1,7
181	0,6	0,8	0	0	2	4	0,7
839	0,6	9,2	0	0	0	4	0,9

Similarly, the task with a start position of (0,6) is assigned to an agent at (2,2), requiring a travel distance of 6 in episode 1. Over a similar period, in episode 181, a task with the same start position is assigned to an agent at (0,7). This improvement is observed again in episode 839, where the task is assigned to an agent at (0,9). The distances of 1 and 3 are significantly lower than the initial distance of 6.

This repeated improvement is observed and supports that the model improves significantly over short periods.

4.3 Variable Look-ahead Parameter, K

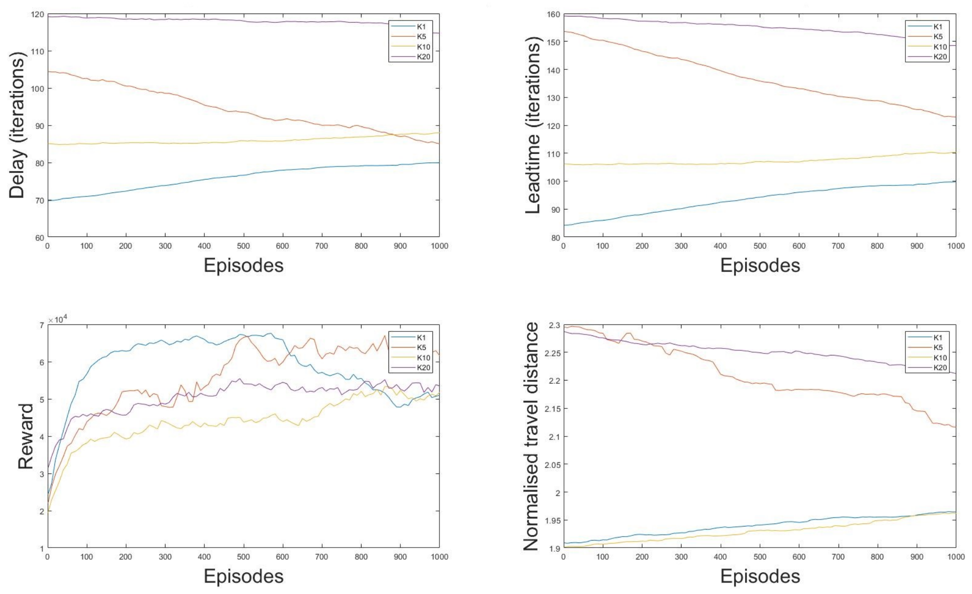


Figure 11 Performance metrics for 4A20T_N0 system

Figure 11 displays the four metrics used to quantify the model's performance on the system; delay, lead time, reward and travel distance.

Overall, all values for the look-ahead parameter show growth in the reward throughout 1000 episodes and the K5 data indicating the "best" performance from a primitive observation.

The data in Figure 12 displays the average reward for the system for a controlled individual queue parameter of $n = 0$, with a varied look-ahead parameter.

Overall, all values of the look-ahead parameter show an increase in reward over the simulation time frame. It can be observed that the reward for the look-ahead value of 1, which indicates no use of the look-ahead functionality, shows the most rapid improvement in the first half of the simulation. However, it peaks and then appears to deteriorate after this point. The initial value of roughly 2.5×10^4 peaks at $\sim 6.8 \times 10^4$ around episode 550 and then drops to a final value of $\sim 5 \times 10^4$.

The remaining look-ahead values show similar growth over the 1000 episodes; however,

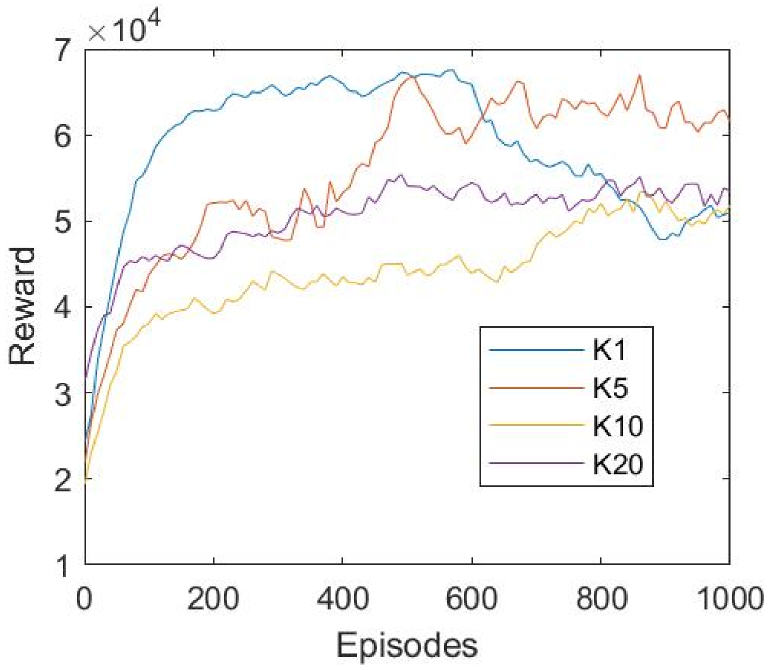


Figure 12 Reward metric for 4A20T_N0 system

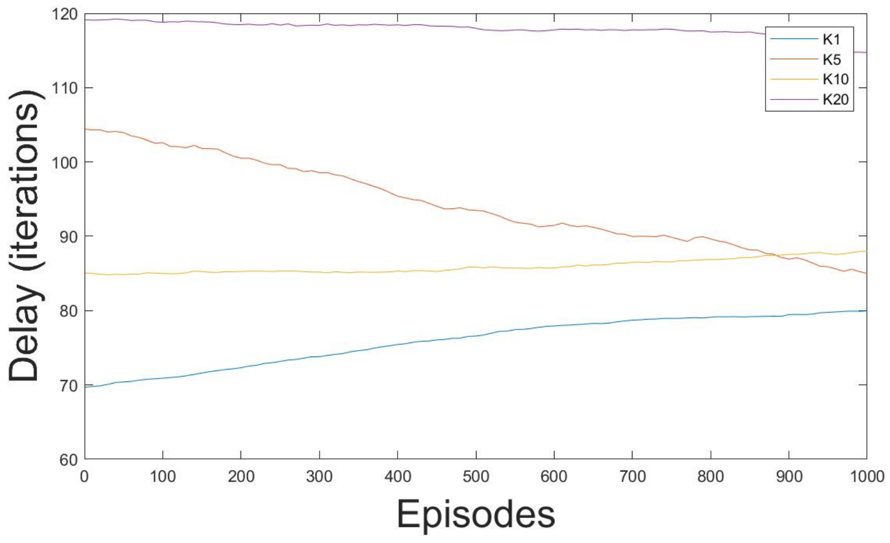
the reward for the systems when the look-ahead parameter is either 5 or 10 result in the most noticeable growth over the simulation. K5 and K10 demonstrate similar growth from $\sim 2.5 \times 10^4$ and $\sim 2 \times 10^4$ to $\sim 6 \times 10^4$ and $\sim 5 \times 10^4$ respectively, exhibiting improvement of over 100% for both systems. Conversely, K20 shows smaller growth from $\sim 3.2 \times 10^4$ to $\sim 5.2 \times 10^4$.

The performance of the varied K values suggests that there is an optimal value for the system, rather than a general trend where increasing it or decreasing it will consistently improve performance. This is supported by numerical evidence of growth, where K1 and K20 perform poorly, while K5 and K10 show tremendous growth. It can be assumed that if the look-ahead value were to increase to greater than 20, the performance would deteriorate or remain at the same level.

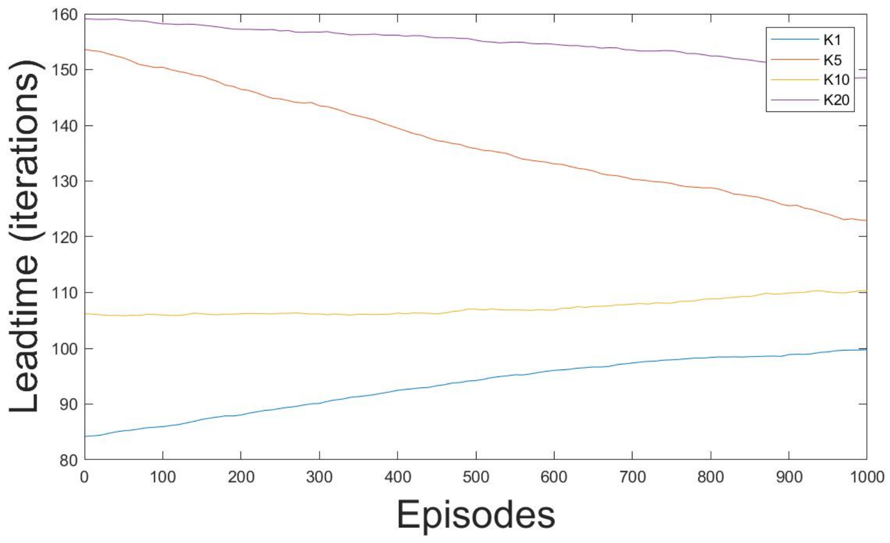
The data in Figure 13 display the average delay and leadtime for the 4A20T_N0 system, with a varied look-ahead parameter.

Overall, there appears to be a trend of an increase in the K value, causing an increase in delay and leadtime. However, the change in delay and leadtime over time is a more significant attribute to note.

The delay for K1 and K10 appears to increase over time, where K10 has an increase of roughly 5% and K1 has an increase of $\sim$15% over 1000 episodes. Conversely, K5 and K20 show a decrease in the average delay over time. The gradient of the K20 data is significantly less steep than that for K5, with a decrease of $\sim$6% compared to the much more significant decrease of just under 20%. It is evident that K5 exhibits the greatest



(a) Delay metric



(b) Leadtime metric

Figure 13 Timing metrics for 4A20T_N0 system

improvement in performance over the course of the experiment.

Due to the similar nature of the metrics, the leadtime values are similar to the delay values. The characteristics of the systems with differing look-ahead values are maintained, where the K5 system demonstrates the most significant improvement, with a decrease of over 20%. The K20 system shows a slight decrease, while the K1 and K10 system shows a slight increase over time.

These increases in the metrics suggest a deterioration in the model's performance; however, this could be attributed to the multi-objective optimisation and the system functioning better under specific parameters. For instance, due to the balance of the reward function, the system could be prioritising the distance travelled rather than the delay and leadtime. However, this is not a significant issue as the problem can be mitigated by adjusting the reward function accordingly.

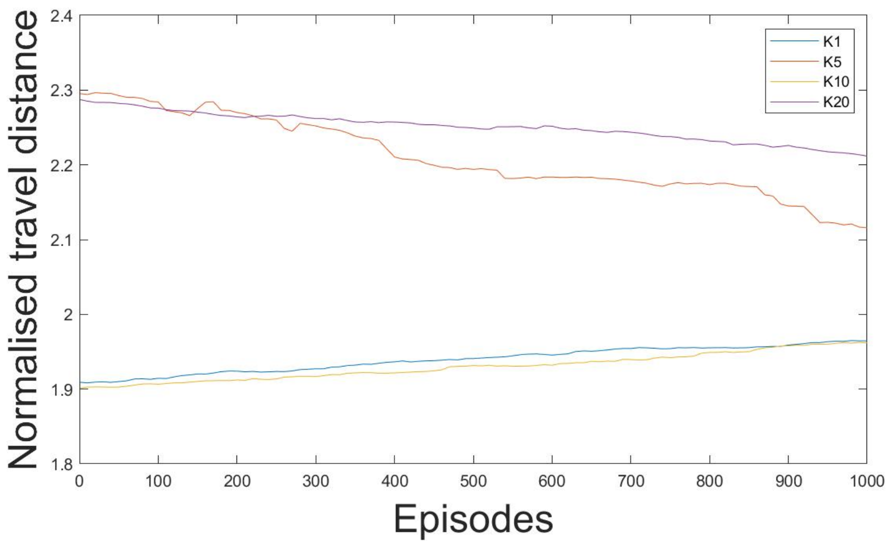


Figure 14 Travel metric for 4A20T_N0 system

The data in Figure 14 and displays the normalised travel distance for the 4A20T_N0 system, with a varied look-ahead parameter.

Similarly to the delay and leadtime, K1 and K10 both show a slight increase of ∼2% in the distance travelled over time. K20 shows repeated behaviour, with a slight decrease of ∼3% compared to that of K5 with a decrease of ∼9% by the end of the experiment.

While K1 and K10 show an increase in the distance travelled, both systems have relatively low initial values for this metric. Therefore, the undesirable behaviour of the K1 and K10 system could be attributed to the inability of this model to converge, given the dynamic nature of the environment and the DQN algorithm on which the model is based. This could be the convergence that is occurring, causing the system to approach the local maximum instead of the local minimum. Another potential cause of this behaviour could be the low initial value causing the system to remain stagnant at its current performance with slight variations, as it deems its performance sufficient.

4.4 Variable Individual Task Queue, N

Figure 15 displays all the metrics for the 4A20T system with a controlled look-ahead parameter of K = 5, and variable individual queue values. Switching the controlled variable is useful to observe the effect of the different variables on the system and gives insight into how to improve the model.

The metrics generally show good performance, with an increase in rewards over time and decreases in delay, leadtime, and travel distance.

Overall, almost all systems show an increase in reward over the simulation time frame;

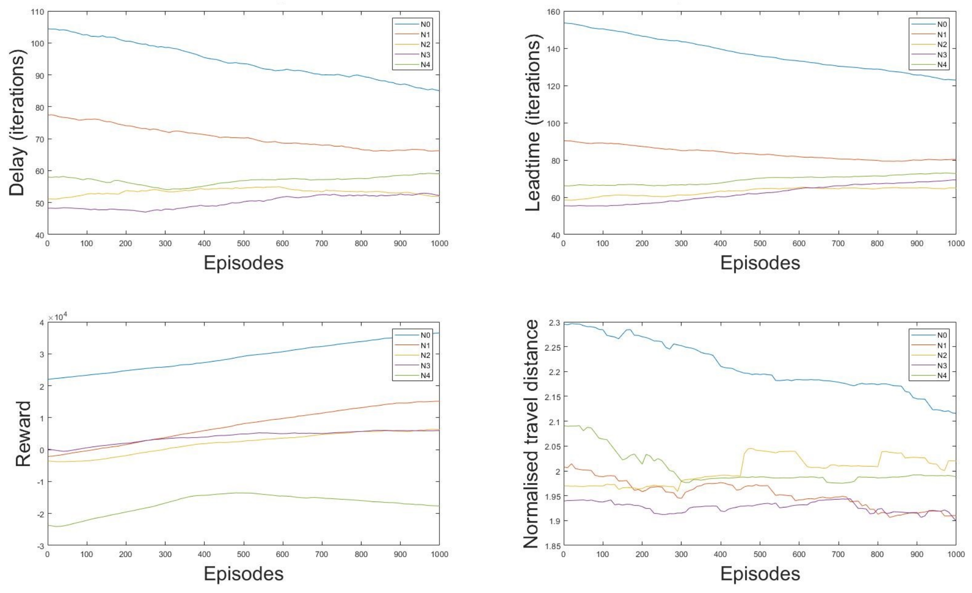


Figure 15 Performance metrics for 4A20T_K5 system

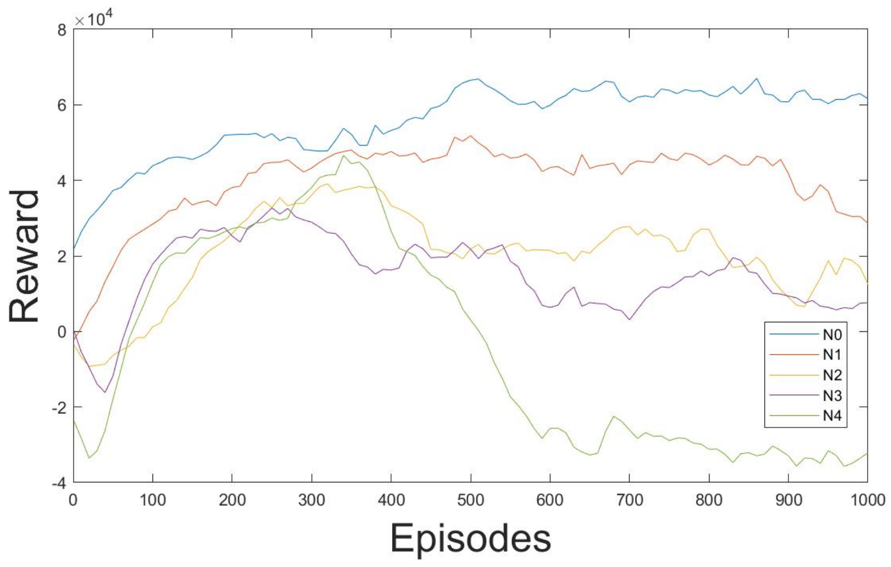


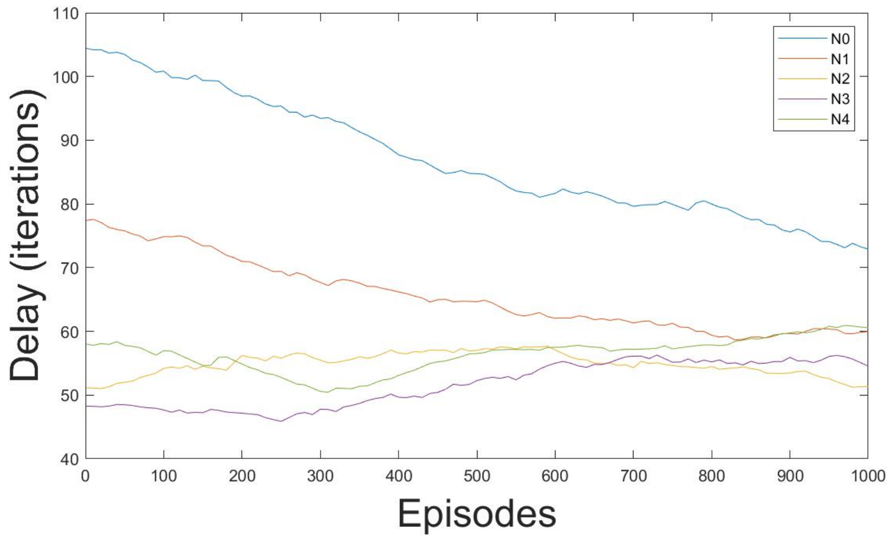
Figure 16 Reward metric for 4A20T_K5 system

however, N4 appears to increase rapidly and decrease below the initial values. A trend can be observed whereby increasing the individual queue causes the reward to decrease.

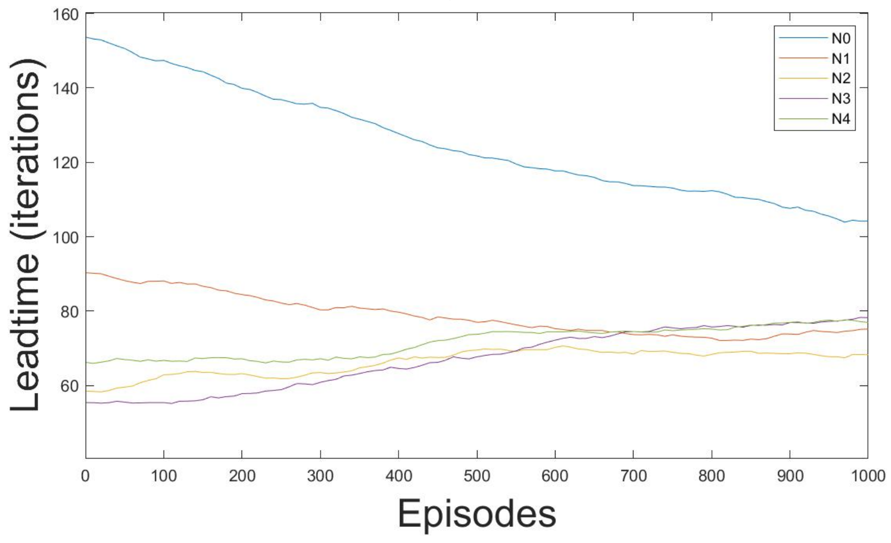
The increase in reward over time tends to decrease with a higher value for the individual queue. In Figure 16 it can be seen that the reward for N0 shows a tremendous increase of $\sim 200\%$ from $\sim 2 \times 10^4$ to $\sim 6 \times 10^4$. N1, N2 and N3 show overall increases of $\sim 3 \times 10^4$, $\sim 1 \times 10^4$ and $\sim 7 \times 10^3$ respectively.

N4 shows a rapid increase in reward from $\sim -2 \times 10^4$ to $\sim 4 \times 10^4$ in roughly 350 episodes. Following this, a rapid decrease to $\sim -3 \times 10^4$ occurs, indicating rapid deterioration of the model's performance over this period.

The trends observed can be attributed to poor task assignment when the queue is long. Additionally, introducing longer queues results in tasks being delayed from execution, and over time this would accumulate a negative reward due to the delay. Furthermore, a shorter queue effectively allows for the dynamic reallocation of tasks compared to the same system with longer individual queues. This is justified as in a system where the agents have a long individual task queue, the tasks assigned to the agent that is deemed most suitable at the time of assignment could be more efficiently assigned to a different agent in the future.



(a) Delay metric



(b) Leadtime metric

Figure 17 Timing metrics for 4A20T_K5 system

As seen in Figure 17, the delay and leadtime metrics do not follow the trend of the rewards, where increasing the agent's queue length does not consistently decrease performance. However, a consistency amongst the reward, delay and leadtime metrics is that N0 and N1 show the most improvement over the 1000 episodes. The leadtime for N0 and N1 decrease by $\sim 32.3\%$ and $\sim 27.8\%$, while the delay decreases by $\sim 28.6\%$ and $\sim 23.1\%$ respectively. The data for N3 has the least desirable progress regarding the timing metrics, where the delay increased by $\sim 12.2\%$, while the leadtime increased by $\sim 36.2\%$. Conversely, N3 has the overall lowest values for delay and leadtime; however, this is not as significant as the progress over time.

N0 and N1 exhibiting the most significant reduction in delay can be attributed to the immediate effect of the exploration-exploitation parameter, ϵ, decreasing over time. As epsilon decays with each episode, the initial task allocation would be random; however, due to the short or non-existent queue, the effect of decaying epsilon on the agents' actions is more evident, thus allowing the considerable reduction visible in the data.

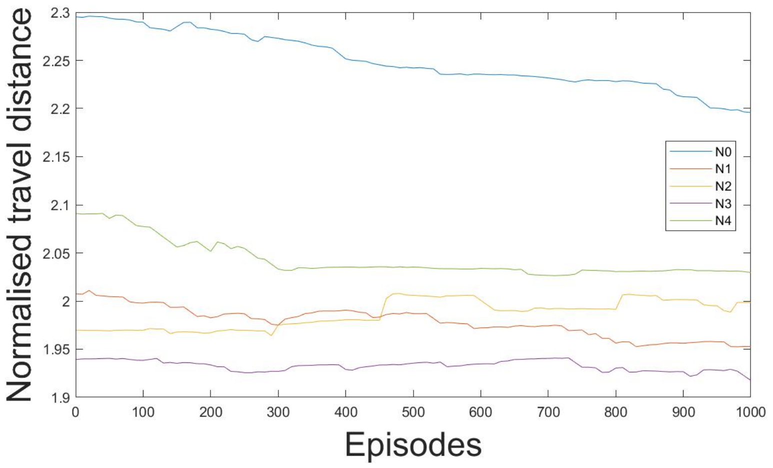


Figure 18 Travel metric for 4A20T_K5 system

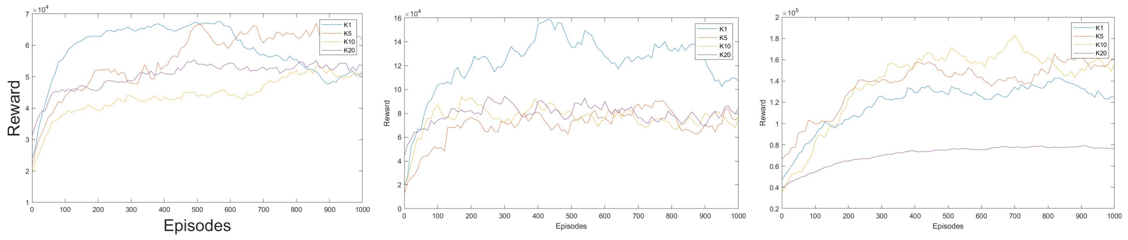
The normalised travel distance metric in Figure 18 does not appear to follow the trend observed in the reward metric, whereby the distance travelled does not consistently increase with an increase in the individual queue value. While N0 showed significant growth in its reward, the distance travelled is the highest compared to the other N values. However, N0 does show the largest reduction in distance travelled of $\sim 4\%$.

N1, N3 and N4 all show similar reductions of $\sim 2.5\%$, $\sim 1\%$ and $\sim 2.9\%$, respectively. Contrarily, N2 shows an increase in distance travelled of $\sim 1.3\%$.

Due to similar behaviour of the different systems in timing metrics and distance travelled, it can be deduced that the causes discussed for the delay and leadtime metrics also apply to the distance travelled metric.

4.5 Comparison of Different Environments

To ensure the robustness of the model, it was tested with various environments and parameters. The performance with different numbers of agents was tested and compared,



(a) 4 agents **(b)** 10 agents **(c)** 20 agents

Figure 19 Average reward for systems with controlled individual queue, N0, with varying number of agents

seen in Figure 19.

Across all three sets of data, it can be observed that the K20 system shows the smallest growth and improvement over time. Conversely, the K1, K5 and K10 system all exhibit significant growth of over 100% in all environments.

The environment with ten agents shows the greatest overall growth numerically, as seen in Table 11. However, the growth of the reward in other environments is consistently high, which suggests an improvement in the model's performance over time, regardless of scaling the environment.

Generally, as the number of agents increases, the numerical value of the reward increases linearly. This is caused by the addition of agents that contribute to the overall reward of the system. However, this is balanced by the negative rewards that are also accumulated with the agents' travel throughout the environment.

Table 11 Numerical progress of reward for different numbers of agents

	4A20T_N0			**10A20T_N0**			**20A20T_N0**		
	Initial	**Final**	**Percentage increase**	**Initial**	**Final**	**Percentage increase**	**Initial**	**Final**	**Percentage increase**
K1	2.5	5	100%	2	10.8	440%	5	12.5	150%
K5	2.2	6.1	177%	1.8	7.8	333%	7	16	129%
K10	2	5	150%	2	7	250%	3.8	15.5	308%
K20	3.1	5.2	68%	4.8	8	67%	4	7.8	95%
Note that all values are x 10^4									

A similarity across the environments with 4 agents and 20 agents is that the optimal performance among the different K values appears to occur at a specific value rather than continuously increasing or decreasing with respect to a particular parameter. This repeated occurrence suggests that the optimal K value varies with the number of agents in the environment.

It is possible that the look-ahead parameter having a value on the same scale as that of the task queue, 20, causes the system to delay or generally perform in an undesirable manner. Since this is a commonality for all three environments in the analysis, it can be deduced that it is caused by varying the look-ahead value, thereby ruling out other potential causes. Furthermore, the behaviour of the look-ahead parameter for scenarios where the length of the task queue is smaller than the look-ahead value, the number of tasks that can be assessed reduces to the number of tasks in the queue at the time. Therefore, in situations where a number of tasks have been completed from the task queue of length 20, the look-ahead parameter effectively decreases. The lack of growth over the episodes could be attributed to this phenomenon.

From this analysis, it is also evident that the number of agents does not significantly affect the model's performance when scaling to different environments.

5. Conclusion

This research has discussed developing and implementing a task allocation model for multi-agent collaborative control of a mobile robot fleet using reinforcement learning. The objectives of this research were to determine the gaps in the existing literature and develop a task allocation model based on these gaps using multi-agent reinforcement learning. The model employs a deep Q-learning algorithm with a reward function that is tailored to facilitate multi-agent reinforcement learning. A novel "look-ahead" parameter was introduced, which takes future tasks into account to encourage efficient task assignment and prevent first-in-first-out behaviour. Simulations were used as the primary form of assessing the model performance conducted on a wide range of devices to ensure its usability. The performance with different value variables such as the number of agents, number of tasks, agent queue length, and the "look-ahead" parameter was assessed. It was determined that the systems with shorter queue lengths generally outperformed those with longer queues. Additionally, it was concluded that the "look-ahead" parameter is required to be tuned and specifically chosen for the particular environment. This task allocation model would be useful in dynamic environments as it promotes the timely completion of tasks and minimal distance travelled between tasks. Significant improvement could be made in the future through further development of the neural network and experimentation on a large autonomous mobile robot fleet.

References

[1] D.-H. Lee, S. A. Zaheer, and J.-H. Kim, “A resource-oriented, decentralized auction algorithm for multirobot task allocation,” *IEEE Transactions on Automation Science and Engineering*, vol. 12, no. 4, pp. 1469–1481, oct 2015.

[2] T. EIJYNE, R. G, and P. S. G, “Development of a task-oriented, auction-based task allocation framework for a heterogeneous multirobot system,” *Sādhanā*, vol. 45, no. 1, may 2020.

[3] T. Srinivasan, V. Vijaykumar, and R. Chandrasekar, “An auction based task allocation scheme for power-aware intrusion detection in wireless ad-hoc networks,” in *2006 IFIP International Conference on Wireless and Optical Communications Networks*. IEEE, 2006.

[4] L. Wang, M. Liu, and M. Q.-H. Meng, “An auction-based resource allocation strategy for joint-surveillance using networked multi-robot systems,” in *2013 IEEE International Conference on Information and Automation (ICIA)*. IEEE, aug 2013.

[5] Y. Zhang and M. Q.-H. Meng, “Comparison of auction-based methods for task allocation problem in multi-robot systems,” in *2013 IEEE International Conference on Robotics and Biomimetics (ROBIO)*. IEEE, dec 2013.

[6] D.-H. Lee, “Resource-based task allocation for multi-robot systems,” *Robotics and Autonomous Systems*, vol. 103, pp. 151–161, may 2018.

[7] S. Wei, D. Lihua, F. Hao, and Z. Haiqiang, “Task allocation for multi-robot cooperative hunting behavior based on improved auction algorithm,” in *2008 27th Chinese Control Conference*. IEEE, jul 2008.

[8] S. N., K. C. R.M., R. M.M., and M. N. Janardhanan, “Review on state-of-the-art dynamic task allocation strategies for multiple-robot systems,” *Industrial Robot: the international journal of robotics research and application*, vol. 47, no. 6, pp. 929–942, sep 2020.

[9] W. P. N. dos Reis and G. S. Bastos, “Multi-robot task allocation approach using ROS,” in *2015 12th Latin American Robotics Symposium and 2015 3rd Brazilian Symposium on Robotics (LARS-SBR)*. IEEE, oct 2015.

[10] D. Goldberg and M. J. Mataric, “Robust behavior-based control for distributed multi-robot collection tasks,” in *Robust Behavior-Based Control for Distributed Multi-Robot Collection Tasks*, 2000.

[11] T. S. Dahl, M. J. Mataric, and G. S. Sukhatme, "Scheduling with group dynamics: A multi-robot task allocation algorithm based on vacancy chains," Center for Robotics and Embedded Systems, Tech. Rep., 2002.

[12] J. Kober, J. A. Bagnell, and J. Peters, "Reinforcement learning in robotics: A survey," *The International Journal of Robotics Research*, vol. 32, no. 11, pp. 1238–1274, aug 2013.

[13] S. Kapetanakis, D. Kudenko, and M. J. A. Strens, "Reinforcement learning approaches to coordination in cooperative multi-agent systems," in *Lecture Notes in Computer Science*. Springer Berlin Heidelberg, 2003, pp. 18–32.

[14] C. Claus and C. Boutilier, "The dynamics of reinforcement learning in cooperative multiagent systems," in *The National Conference on Artificial Intelligence (AAAI 1998)*, 1998.

[15] J. Hao, D. Huang, Y. Cai, and H. fung Leung, "The dynamics of reinforcement social learning in networked cooperative multiagent systems," *Engineering Applications of Artificial Intelligence*, vol. 58, pp. 111–122, feb 2017.

[16] I. Santín, C. Pedret, and R. Vilanova, *Control and Decision Strategies in Wastewater Treatment Plants for Operation Improvement*. Springer International Publishing, 2017.

[17] F. Jiang, W. Liu, J. Wang, and X. Liu, "Q-learning based task offloading and resource allocation scheme for internet of vehicles," in *2020 IEEE/CIC International Conference on Communications in China (ICCC)*. IEEE, aug 2020.

[18] M. Andrychowicz, F. Wolski, A. Ray, J. Schneider, R. Fong, P. Welinder, B. McGrew, J. Tobin, P. Abbeel, and W. Zaremba, "Hindsight experience replay," *Advances in Neural Information Processing Systems 30 (NIPS 2017)*, 2017.

[19] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, "Playing atari with deep reinforcement learning," *arXiv*, 2013.

[20] W. Dai, H. Lu, J. Xiao, Z. Zeng, and Z. Zheng, "Multi-robot dynamic task allocation for exploration and destruction," *Journal of Intelligent & Robotic Systems*, vol. 98, no. 2, pp. 455–479, oct 2019.

[21] A. Malus, D. Kozjek, and R. Vrabič, "Real-time order dispatching for a fleet of autonomous mobile robots using multi-agent reinforcement learning," *CIRP Annals*, vol. 69, no. 1, pp. 397–400, 2020.

[22] C. E. Pippin and H. Christensen, "Learning task performance in market-based task allocation," in *Advances in Intelligent Systems and Computing*. Springer Berlin Heidelberg, 2013, pp. 613–621.

[23] A. Steinbach, "Actor-critic using deep-rl: continuous mountain car in tensorflow," Available at https://medium.com/@asteinbach/actor-critic-using-deep-rl-continuous-mountain-car-in-tensorflow-4c1fb2110f7c (2020/06/12).

[24] J. Feng, F. R. Yu, Q. Pei, X. Chu, J. Du, and L. Zhu, "Cooperative computation offloading and resource allocation for blockchain-enabled mobile-edge computing: A deep reinforcement learning approach," *IEEE Internet of Things Journal*, vol. 7, no. 7, pp. 6214–6228, jul 2020.

[25] M. Ahrarinouri, M. Rastegar, and A. R. Seifi, "Multiagent reinforcement learning for energy management in residential buildings," *IEEE Transactions on Industrial Informatics*, vol. 17, no. 1, pp. 659–666, jan 2021.

[26] K. Zhang, Z. Yang, H. Liu, T. Zhang, and T. Basar, "Fully decentralized multi-agent reinforcement learning with networked agents," *arXiv*, 2018.

[27] B. C. Wagstaff and F. O'Brien, *Development of Autonomous Mobile Robots for Industry 4.0 Smart Factories*. The University of Auckland, 2020.

[28] M. J. Matarić, G. S. Sukhatme, and E. H. Østergaard, "Multi-robot task allocation in uncertain environments," *Autonomous Robots*, vol. 14, no. 2/3, pp. 255–263, 2003.

[29] Z. Liu, M. Ang, and W. Seah, "Reinforcement learning of cooperative behaviors for multi-robot tracking of multiple moving targets," in *2005 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, 2005.

[30] J. Fan, M. Fei, L. Shao, and F. Huang, "A novel multi-robot coordination method based on reinforcement learning," in *Lecture Notes in Computer Science*. Springer Berlin Heidelberg, 2008, pp. 1198–1205.

[31] J. Shao, J. Zhang, and C. Zhao, "Research on multi-robot path planning methods based on learning classifier system with gradient descent methods," in *Advances in Intelligent and Soft Computing*. Springer Berlin Heidelberg, 2012, pp. 229–234.

[32] J. Turner, "Distributed task allocation optimisation techniques," in *International Foundation for Autonomous Agents and Multiagent Systems*. International Foundation for Autonomous Agents and Multiagent Systems, 2018.

[33] G. P. Das, T. M. McGinnity, S. A. Coleman, and L. Behera, "A distributed task allocation algorithm for a multi-robot system in healthcare facilities," *Journal of Intelligent & Robotic Systems*, vol. 80, no. 1, pp. 33–58, nov 2014.

[34] S. Nayak, S. Yeotikar, E. Carrillo, E. Rudnick-Cohen, M. K. M. Jaffar, R. Patel, S. Azarm, J. W. Herrmann, H. Xu, and M. Otte, "Experimental comparison of decentralized task allocation algorithms under imperfect communication," *IEEE Robotics and Automation Letters*, vol. 5, no. 2, pp. 572–579, apr 2020.

[35] L. Jin and S. Li, "Distributed task allocation of multiple robots: A control perspective," *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, vol. 48, no. 5, pp. 693–701, may 2018.

[36] K. Zhang, Z. Yang, and T. Basar, "Multi-agent reinforcement learning: A selective overview of theories and algorithms," *ArXiv*, 2019.

[37] M. Lauer and M. Riedmiller, "An algorithm for distributed reinforcement learning in cooperative multi-agent systems," in *In Proceedings of the Seventeenth International Conference on Machine Learning*, 2000.

[38] R. Lowe, Y. Wu, A. Tamar, J. Harb, P. Abbeel, and I. Mordatch, "Multi-agent actor-critic for mixed cooperative-competitive environments," *arXiv*, 2020.

[39] B. Yongacoglu, G. Arslan, and S. YÃŒksel, "Decentralized learning for optimality in stochastic dynamic teams and games with local control and global state information," *arXiv*, 2021.

[40] T. Luo, B. Subagdja, D. Wang, and A.-H. Tan, "Multi-agent collaborative exploration through graph-based deep reinforcement learning," in *2019 IEEE International Conference on Agents (ICA)*. IEEE, oct 2019.

[41] M. Otte, M. J. Kuhlman, and D. Sofge, "Auctions for multi-robot task allocation in communication limited environments," *Autonomous Robots*, vol. 44, no. 3-4, pp. 547–584, jan 2019.

[42] M. Roshanzamir, M. A. Balafar, and S. N. Razavi, "A new hierarchical multi group particle swarm optimization with different task allocations inspired by holonic multi agent systems," *Expert Systems with Applications*, vol. 149, p. 113292, jul 2020.

[43] J. Han, Z. Zhang, and X. Wu, "A real-world-oriented multi-task allocation approach based on multi-agent reinforcement learning in mobile crowd sensing," *Information*, vol. 11, no. 2, p. 101, feb 2020.

[44] Z. Tong, X. Deng, H. Chen, J. Mei, and H. Liu, "QL-HEFT: a novel machine learning scheduling scheme base on cloud computing environment," *Neural Computing and Applications*, vol. 32, no. 10, pp. 5553–5570, mar 2019.

[45] X. Deng, J. Li, E. Liu, and H. Zhang, "Task allocation algorithm and optimization model on edge collaboration," *Journal of Systems Architecture*, vol. 110, p. 101778, nov 2020.

[46] K. Li, T. Zhang, and R. Wang, "Deep reinforcement learning for multi-objective optimization," *arXiv*, 2019.

[47] H. nan Wang, N. Liu, Y. yun Zhang, D. wei Feng, F. Huang, D. sheng Li, and Y. ming Zhang, "Deep reinforcement learning: a survey," *Frontiers of Information Technology & Electronic Engineering*, vol. 21, no. 12, pp. 1726–1744, oct 2020.

[48] B. Abdualgalil and S. Abraham, "Applications of machine learning algorithms and performance comparison: A review," in *2020 International Conference on Emerging Trends in Information Technology and Engineering (ic-ETITE)*. IEEE, feb 2020.

[49] J. Zhu, J. Shi, Z. Yang, and B. Li, "A real-time decentralized algorithm for task scheduling in multi-agent system with continuous damage," *Applied Soft Computing*, vol. 83, p. 105628, oct 2019.

[50] Z.-Y. Zhang, Y.-F. Chen, D. Xu, S.-Y. Gong, and Y.-L. Meng, "Research on multi-robot task allocation algorithm based on HADTQL," in *2020 International Workshop on Electronic Communication and Artificial Intelligence (IWECAI)*. IEEE, jun 2020.

[51] D. B. Noureddine, A. Gharbi, and S. B. Ahmed, "Multi-agent deep reinforcement learning for task allocation in dynamic environment," in *Proceedings of the 12th International Conference on Software Technologies*. SCITEPRESS - Science and Technology Publications, 2017.

[52] Y. Ishihara and T. Sugawara, "Multi-agent task allocation based on the learning of managers and local preference selections," *Procedia Computer Science*, vol. 176, pp. 675–684, 2020.

[53] M. Li, Z. Wang, K. Li, X. Liao, K. Hone, and X. Liu, "Task allocation on layered multi-agent systems: When evolutionary many-objective optimization meets deep q-learning," *IEEE Transactions on Evolutionary Computation*, pp. 1–1, 2021.

[54] A. Koeller, "Evaluation of reinforcement-learning algorithms in the context of autonomous vehicles," Ph.D. dissertation, RWTH Aachen University, 2019.