Predator-Prey Q-learning for Swarm Coverage Path Planning

Student Name: Michael Watson

Supervisor Name: Junyan Hu

Submitted as part of the degree of BSc Computer Science to the

Board of Examiners in the Department of Computer Sciences, Durham University

Abstract — Coverage Path Planning (CPP) is a problem in which an agent aims to cover an area by finding an efficient path through the environment. In this paper, we focus on using machine learning to learn about a given environment and find the best path which explores each location in the environment. This paper aims to overcome the problems caused by Q-learning based coverage path planning, which is that they often fall into a local optimum. This paper will explore two methods of improvement, one is the use of a swarm which contains multiple agents all working towards the same goal. The other method explored is the augmenting of the Q-learning algorithm to include a predator-prey based reward system. Existing predator-prey based reward systems provide rewards the further away an agent is from its predator, the paper adapts the existing method to work within a swarm by simulating each agent of the swarm as both predator and prey. Simulation results and comparisons between the standard Q-learning and the Predator-Prey based Q-learning show that the improved method performs much better in complicated environments.

Index Terms — Machine Learning, Robotics, Multi-Agent Systems

1 Introduction

THE coverage of unknown environments is a very interesting and important field; it is ever-growing and now with the rise of machine learning there is the possibility for it to be done autonomously. This can be done with the formation of a coverage task such as coverage path planning (CPP). This task aims to completely cover the entirety of an environment which means the agent must each location [1]. There are many uses for coverage tasks, such as search and rescue [2], or the surveying of unexplored locations [3], [4]. They have important impacts, as they can help save people in the case of search and rescue or be used to explore unknown territories. Exploration in this form is especially useful as the use cases can range from a room to a vast wilderness.

There exist several different methods, but this paper aims to improve upon one of the most known reinforcement learning algorithms, Q-learning [5], [6]. Q-learning is a long-standing algorithm and is popular due to its simple implementation allowing it to be easily adapted to several situations. However, this simpleness also leads to several limitations such as easily falling into a local optimum or struggling in more complex environments. These are the main limitations the solution aims to overcome; it is important to overcome these issues as in more complex use cases such as exploring dangerous terrain and environments it is vital that the agents can find optimal paths to save on resources. A shorter path means that more of an environment can be explored, both in total and within a given period. Furthermore, the algorithm must be good in more complex environments where there may exist obstacles that the agents must avoid as it is extremely unlikely that in real-world use cases, the environment will be unobstructed. Another key technology used throughout this paper is swarm robotics [7], swarms are a collection of multiple robots all working together towards the same goal. They are highly effective in exploration tasks as they multiply how much of the environment can be explored at once. Therefore, this paper focuses on implementing the upgraded algorithm into a swarm so that they can be used together to provide a superior method of exploration.

The paper aims to answer the question: "Can you adapt Q-learning to be more effective in Coverage Path Planning problems?"

To attempt to answer this question the paper provides a completed algorithm and explains the method used to obtain the results displayed at the end of the paper. It will explain the Q-learning method used as well as the adaptions made to improve it for the necessary purpose. A complete program was made to test the algorithms against each other simulating custom environments, as well as easily allowing differing swarm sizes. An implementation of a standard algorithm was also implemented to be compared against. The adapted solution aims to improve upon the existing method in several ways, it aims to improve performance in complex scenarios in which obstacles exist increasing the difficulty of exploration. Another vital result is to ensure that full coverage always occurs within the given movement budget, it also aims to improve the efficiency of the algorithm by reducing the average number of steps needed to fully cover. Finally, the solution aims to be consistent so that it pro-

duces reliable results.

2 Related Work

There exist several CPP algorithms designed for mobile robots. The task is important for many robotic applications such as vacuum cleaning robots [8], [9], lawnmowers [10], [11], [12] and automated harvesters [13], [14]. One early paper on CPP [1] defines the requirements an agent must meet to perform a coverage. The requirements are:

1. The mobile robot must move through an entire area
2. The mobile robot must fill the region without overlapping paths.
3. Continuous and sequential operation without any repetition of paths is required.
4. The robot must avoid all obstacles in the environment.
5. Simple motion trajectories (e.g., straight lines or circles) should be used for simplicity.
6. An "optimal" path is desired under available conditions.

It is not always possible to satisfy all criteria in complex environments so sometimes a priority consideration is required. The CPP problem shares similarities with other existing problems such as the travelling salesman problem (TSP) [15] or the lawn mowing problem [16]. Although these problems differ slightly such as the TSP requires visits to each main location such as a city, whereas CPP requires visiting every location such as each street or neighbourhood. Furthermore, neither the TSP nor the lawn mowing problem account for obstacles in the path of the agent.

There are multiple solutions to the CPP problem which can be categorized into offline and online algorithms [17]. Offline algorithms perform coverage in a static and known environment, whereas online algorithms work in dynamic environments. The online algorithm is preferred as most use cases will involve dynamic environments in which the agent will have to adapt. However, even online algorithms have their limitations one being they require agents with sensors to learn about their surroundings. Another is that they lack the same efficiency as offline algorithms which can find an optimal path and then stick to it. CPP algorithms can also take many different approaches [18] such as machine learning [19], greedy search [20], graph search [21], frontier-based exploration [22] as well as many others. Each method has its strengths and weaknesses and the best method to use often depends on the use case. For example, the greedy algorithm (Dijkstra's algorithm) is simple, easy to implement and generally fast, but it does not guarantee a global optimum. It is also an offline algorithm which means it cannot be used in dynamic environments.

Machine learning (ML) is the method which this paper focuses on, there are several different possible ML algorithms such as reinforcement learning (RL) and deep learning. RL algorithms allow the agent to learn to reach a desired goal using actions and rewards [23]. In an RL algorithm, the agent selects an action from all available actions based on what it predicts the reward given from that action will be. The actual reward is then given to the agent using the determined reward structure. The reward structure is a key part of the algorithm and can determine the effectiveness of a given solution. Therefore, this is often the limitation of a given solution, with a better reward structure resulting in better results.

Piardi et al. [24] present a Q-learning algorithm which uses a grid-based map to optimize the CPP trajectory, this implementation has some limitations. Its simple nature means it doesn't work in environments with unknown obstacles. However, it does provide a good basis for improvement by adapting this algorithm with further methods. Zhou et al. [25] provide numerous improvements over a standard Q-learning algorithm to create an optimized algorithm, optimized Q-learning. It includes improvements such as novel Q-table initialization, a new action-selection policy, and a new reward function. The limitations of this paper however are that its reward function is still simplistic, and it also lacks integration for multiple robots. Puente-Castro et al. [26] make use of a swarm of agents to accomplish the task as a group. This method effectively demonstrates the usefulness of a swarm over an individual agent, it shows how using a swarm reduces the individual load of each agent whilst maintaining the overall result of the solution. The limitation of this method however is the reward structure, it is very simple and therefore leaves room for improvement. One method to improve the reward structure is through the use of a predator-prey based reward structure [27]. This reward structure combines three different reward functions to produce a reward for the agent. These rewards given are the predator-prey reward, a boundary reward, and a smoothness reward, in combination this provides the agent with a much more responsive representation of its environment. The limitation of this method comes from its lack of swarm implementation, it focuses on only one agent and with multiple agents, collisions can reduce the effectiveness of the rewards. It must therefore be adapted for a swarm.

3 Methodology

This solution aims to create an efficient reinforcement learning algorithm for use in a coverage path planning scenario. Robot agents are used to complete the coverage, they have a limited action set being able to only move forward, back, left, and right. They were utilized in a swarm to increase the speed at which the entirety of an environment can be covered, the advantages to using a swarm are plentiful. The multiple agents mean more of the environment can be explored concurrently; it also provides redundancies in the scenario that an agent is lost. However, there are some drawbacks in the way that the swarm interacts such as collisions as well as issues in how the swarm explores. The solution to this is to adapt a reinforcement learning algorithm around the swarm to overcome these issues.

The reinforcement learning method Q-Learning was selected as it is an easy to implement method which can

be used easily in a swarm as if needed the agents can act individually. It does experience some limitations when in a swarm as the basic implementation has structure when in a swarm and does not handle the interaction between agents, this is an issue solved with the adapted solution.

3.1 Foundation Work

An implementation of Watkins Q-learning [5] was used for this solution. Q-learning is a simple reinforcement learning algorithm which makes use of a data structure called a Q-table and a function known as the Q function. Q learning works by initializing the agent's Q-table (Table 1) which holds a predicted reward for each state the agent can be in, and each action it can take from this state. The estimated rewards are then continually improved using a specialized Q-Function (1) to produce rewards which represent how beneficial each action is. An example of the improved values can be seen in Table 2, some values remain 0 as these are invalid moves such as moving out of the environment.

Table 1:
EXAMPLE OF INITIALIZED Q-TABLE

		Actions			
		Up (0)	Down (1)	Left (2)	Right (3)
States	0	0	0	0	0
	1	0	0	0	0
	.	.	.	.	.
	.	.	.	.	.
	.	.	.	.	.
	N	0	0	0	0

Table 2:
EXAMPLE OF IMPROVED Q-TABLE

		Actions			
		Up (0)	Down (1)	Left (2)	Right (3)
States	0	0.123	0	0	0.721
	1	0.545	0	0.103	0.873
	.	.	.	.	.
	.	.	.	.	.
	.	.	.	.	.
	N	0	0.391	0.654	0

The function takes the observed reward and the estimated future value and updates the value in the table according to the function. The function is formulated as follows:

$$Q(s,a) = Q(s,a) + \alpha . [r + \gamma . max\, Q(s`) - Q(s,a)] \quad (1)$$

where $Q(s,a)$ represents the current Q-value for the given state and action, $s`$ refers to the next state (the state the current action will lead to), $\alpha = Learning\ Rate, r = reward$ and $\gamma = Discount\ Factor$. The function can be altered through the use of the parameters α, γ. These values can range from 0 to 1. The learning rate determines how much each learning increment will affect the Q-Value, the greater the learning rate the faster the Q-Values will converge. It is therefore important to have a learning rate small enough that it doesn't converge too early but also ensure that it's not so small it takes too long to converge. The discount factor determines how much importance is placed on the future reward. The greater the discount factor the more important the function finds the future reward; it is also important to balance this parameter so that the correct amount of weighting is placed on the potential future.

Another significant strategy implemented was the Epsilon-Greedy method. This is a method of introducing some randomness into the agent's selection of an action, which allows it to explore and find better paths and actions it can use. It works by using a value denoted as epsilon, the agent will then choose a random action with probability ε and the best action with probability $1 - \varepsilon$. The value of epsilon then decays and reduces after each cycle so fewer random actions are taken. Performance can be heavily impacted by both the starting value and the decay rate of epsilon. If epsilon remains too high the algorithm will take too many random actions and never be able to converge on a suitable answer. However, if epsilon starts to low, no random actions are taken, and the agent gets stuck in a local optima never branching off to explore potentially better alternatives.

A combination of these methods can be seen in Algorithm 1

Algorithm 1: Q-learning

```
Initialize Q(s, a) as 0
Loop for each episode:
    Initialize s
    Loop for each step of the episode:
        Random p
        If p < ε:
            Any action a
        Else:
            a = Max Q(s)
        Take action a, gather r, s`
        Q(s,a) = Q(s,a) + α.[r + γ.max Q(s`) − Q(s,a)]
        s = s`
    Until s is terminal
```

The basic reward structure used for standard Q-learning is simple, a positive reward is given the first time a location is seen, and a negative reward is given each time the agent moves. This negative reward acts as a movement penalty, it is given so that the agent will aim to cover the environment in a minimal number of steps as the more it moves the smaller the reward it gains.

$$r = \begin{cases} 1, & Undiscovered\ Location \\ -0.05, & Movement\ Penalty \end{cases}$$

Backtracking is another method implemented into the solution to assist the agent in what to do when it sees no unvisited neighbours. If the agent reaches a position where it has no unvisited neighbours, it will begin to retrace its steps until it sees an unvisited location and it will then return to normal. The use of backtracking is to overcome situations where the agent becomes stuck in a loop moving between locations it has already seen until it ter-

minates. Although backtracking is not efficient it is effective at ensuring the environment will always be completely covered and the agent will not get stuck. Coverage is guaranteed as anywhere the agent can reach it must have passed at some point and this method of backtracking ensures it is seen again and can be explored this time. This does create some limitations however as this form of backtracking is not very efficient because it retraces its steps. This occurs each time the agent starts backtracking. It could be improved by implementing a better method which remembers the location where the agent stopped backtracking and returned to exploration, this location could then be returned if the agent needs to backtrack again.

3.2 Swarm Structure

This solution makes use of multiple agents working together to create a centralized swarm. The implementation of the swarm allows for the use of a changing number of agents, the swarm can contain any number of agents and still function correctly. Using more agents will allow for faster exploration, but it requires greater resources. The agents of the swarm all use the same algorithm however they use individual Q-tables so they can determine their best paths individually (Figure 1). This is necessary to take full advantage of the swarm as with a shared Q-Table all agents would attempt to follow the same path which would negate the advantages of using a swarm in the first place. The use of one Q-table would also lead to agents disrupting other agents' Q-values leading to worse results than using a single agent.

There is also a central controller of the swarm which holds information about the swarm. This includes information about the locations of all swarm members, which allows any member to know the position of the other members. This can be used for both avoiding collisions as well as in the reward function (2). The controller also holds how much coverage the swarm has completed so it can monitor progress and tell the members to stop once coverage is complete.

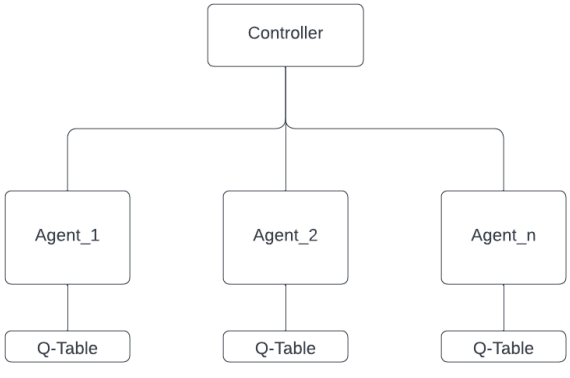

Figure 1 - Example of the Swarm Structure

There also exist some potentially problematic interactions with agents in the swarm, one of which is how the agents avoid crashing. In the simulation, a crash occurs when two agents move into the same location. This is overcome by restricting the agent's movement to not allow actions which will result in the agents overlapping. This can reduce performance sometimes as agents may have to wait for other agents to move first before they can, but it is vital to ensure all agents remain safe.

Another problem is the fact that with multiple agents exploring different locations, the agents cannot know when they have collectively covered the entirety of the environment. Without some shared information of what each agent has covered the agents will run until they have covered the entire environment individually. This is overcome using a simulated server that holds the amount of the environment covered, this allows any agent to access the collective memory to share what they have covered and know when to stop. The server also holds the specific locations that have been explored, this avoids duplicate counting as well as allowing the correct rewards to be given to the agent for unexplored and explored locations.

3.3 Predator-Prey Improvement

Predator Prey is a method which simulates the animal relationship between predator and prey. Specifically, it mimics the behaviour of the prey, in which they aim to get as far from the predator as quickly as possible. This method was adapted into a reinforcement learning scenario by Zhang et al. [27]. The method implemented in their paper focuses on using one agent to fully explore an environment. This is done by simulating a predator at one end of the environment so that the agent will move quickly away from it, the method is effective at getting the agent to move in straight lines as it is its optimal pathing for maximum distance. This behaviour is especially useful in a coverage scenario as moving in straight lines avoids any small pockets of unvisited locations being missed and needing to be found later. Once a boundary is reached the predator can then be moved to encourage the agent to move in a direction of choice, by doing this repeatedly the agent can sweep the entirety of the environment very effectively.

The solution is built upon this method; however, it is altered to work in a swarm. The key difference is that instead of simulating a predator for the agent to run from all agents in the swarm simulate both predator and prey. An individual agent aims to move as far as possible from the other agents in the swarm, simulating the prey with every other member of the swarm acting as its predator. This method is effective when the entire swarm begins in the same region of the environment as it means the swarm spreads out which allows for different regions of the environment to be explored faster. A reward is given depending on how far the agent is from its closest predator (2). The reward for moving to neighbour x is formulated as follows:

$$r_p(x) = \frac{D(x) - D_{min}(x_n)}{D_{max}(x_n) - D_{min}(x_n)} \quad (2)$$

where $D(x)$ gives the distance from x to its nearest predator. $D_{max}(x_n)$ gives the maximum distance between one of x's neighbours and any predator, and $D_{min}(x_n)$ gives the minimum distance between one of x's neighbours and any predator.

Two further functions are also used which give rewards based on the smoothness of the path travelled and the boundary of the next state. The smoothness reward gives a reward based on the direction of the agent's travel (3), if it moves in the same direction multiple times it receives a greater reward. This is done to entice the agent into moving in straight lines, as well as discourage moving back along its path when unnecessary. The reward for moving to neighbour x is formulated as follows:

$$r_s(x) = \begin{cases} \angle\, x_{k-1}x_k x = 0, & reward = 1 \\ \angle\, x_{k-1}x_k x = 90, & reward = 0.5 \\ \angle\, x_{k-1}x_k x = 180, & reward = 0 \end{cases} \quad (3)$$

where x_{k-1} and x_k are the positions covered at k-1 and k, respectively. The reward is given based on the angle between the vectors $(x_{k-1} - x_k)$ and $(x_k - x)$.

The boundary reward gives a reward based on the number of unvisited neighbours a given location has (4). The fewer the neighbours the greater the reward. This helps the agent to move into locations which may otherwise get cut off and left behind. The reward for moving to neighbour x is formulated as follows:

$$r_b(x) = \frac{N_{max} - x_N}{N_{max}} \quad (4)$$

where N_{max} is the total number of neighbours and x_N is the number of unvisited neighbours x has.

The rewards are then combined to give a complete reward. They are combined with different weights to manipulate the importance of each individual reward. It is formulated as follows:

$$r(x) = w_p\left(r_p(x)\right) + w_s\left(r_s(x)\right) + w_b\left(r_b(x)\right) \quad (5)$$

where $w_p w_s w_b$ represent the weights for the predation reward, smoothness reward and boundary reward, respectively. $r(x)$ is the final reward which is returned to the agent. The weights can be changed to differing values to change how rewards are given to the agent. A larger weight means more emphasis is placed on that reward; this allows for the algorithm to work in different ways such as focusing on spreading out as quickly as possible or trying to move in only straight lines. Using powerful values for the weights which are much higher than the others can lead to a worse performance as it overpowers the other reward functions and only one is truly considered.

The pseudocode for the complete algorithm can be seen in Algorithm 2. This combines all improvements including the use of a swarm, the use of backtracking and the use of the adapted reward function.

Algorithm 2: Improved Predator-Prey Swarm Q-learning

```
Initialize Swarm
For each agent in Swarm:
    Initialize Q(s, a) as 0
Initialize Exploration_Count
Loop for each episode:
    For each agent in Swarm:
        Initialize s
    Loop for each step of the episode:
        For each agent in Swarm:
            Random p
            If p < ε:
                Any action a
            Else:
                If there is an unvisited location:
                    a = Max Q(s)
                Else:
                    a = Previous Action
            Take action a, gather s`, gather r using (5)
            Q(s,a) = Q(s,a) + α.[r + γ.max Q(s`) − Q(s,a)]
            If s` is unvisited:
                Exploration_Count += 1
            s = s`
    Until Exploration_Count is terminal
```

There are some constraints to consider when using this solution such as it only considers a static 2D environment and is therefore limited in its applications. As well as the scalability of the solution is dependent on how much memory is available. This is because Q-learning requires a large amount of memory to use as each agent requires its own Q-table; the size of the Q-table can become very large as it requires an entry for every action at each location in the environment. This can be overcome by splitting the environment into smaller subsections and ensuring a simple action set for the agents such as: forward, backwards, left and right.

4 Results

The results are simulated in a custom-made environment, the environment is a square grid which can vary in size. The grid also can include obstacles in the environment to make it more complicated to explore. The obstacles are also completely customizable and can vary between different simulations, to allow testing to be more varied.

The program will be compared with obstacles, different swarm sizes and against an implementation of a standard Q-swarm algorithm.

Throughout the testing the weights used in (5) are $w_p = 2, w_s = 1, w_b = 1.5$. The learning rate was 0.001, the discount factor was 0.9, the starting epsilon value was 0.2 and there were 1000 learning episodes carried out. There was a movement budget set based on the environment size, this was the maximum number of steps allowed for each coverage attempt.

In this results section there are three types of plots, trajectory, coverage/time, and movement/time. In this case

of coverage/time and movement/time 50 simulation loops are run to get an average and accurate result. Plots contain several lines each one representing one of the simulations run, there often appear to be fewer lines as they overlap in most cases.

The trajectory graph shows the path of each swarm member through the coverage showing which location was covered by which agent.

The coverage/time graphs show the percentage of the environment covered during a certain period. This allows you to see how quickly the swarm covers the environment as well as if it successfully covers the environment.

The movement/time graph shows how many steps the swarm took to cover the environment. This allows you to see the efficiency of the swarm, the fewer the steps taken the more efficient the algorithm was.

The following plots illustrate an example of coverage in an environment of size 12 with a swarm size of 5 agents. There were no obstacles in this environment.

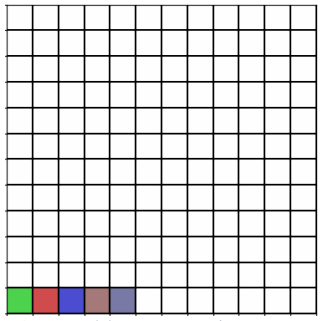

Figure 2(a) – 0 seconds

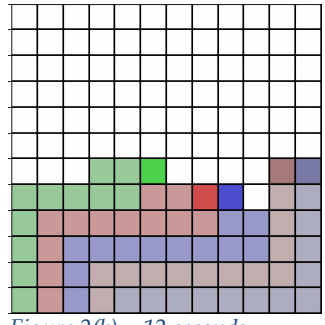

Figure 2(b) – 12 seconds

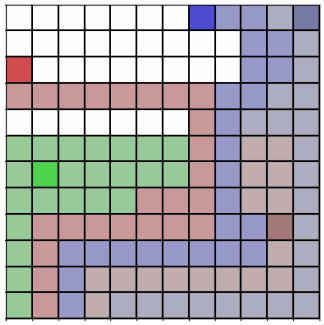

Figure 2(c) – 24 seconds

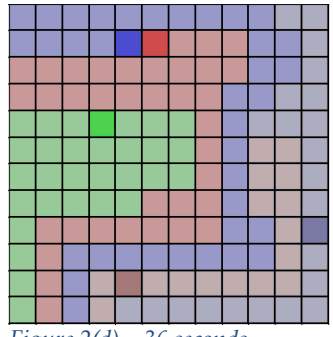

Figure 2(d) – 36 seconds

Figure 2 – This is the trajectory over time of the Predator-Prey Algorithm with swarm size 5 on an environment size 12 with no obstacles. It shows the path each robot took to cover.

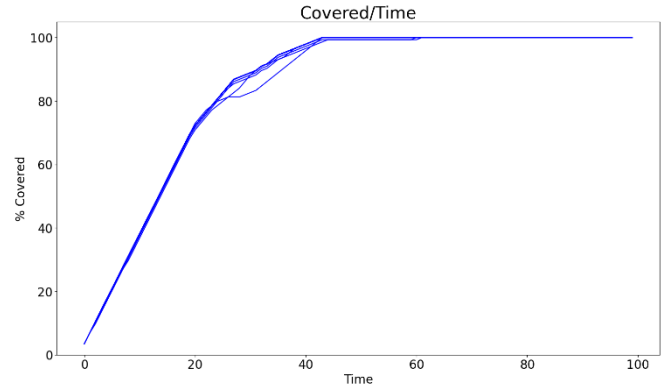

Figure 3 – The Coverage/Time of a Predator-Prey Algorithm with swarm size 5 on an environment size 12 with no obstacles. It shows the consistency of the predator-prey algorithm.

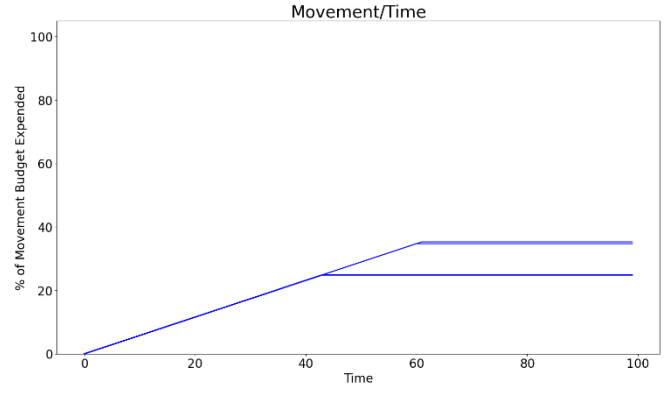

Figure 4 - Movement/Time of a Predator-Prey Algorithm with swarm size 5 on an environment size 12 with no obstacles. This shows the efficiency of the algorithm.

These figures show an example of coverages in the same environment as the previous plots (size of 12 with 5 agents), but there are obstacles in this environment. These results will show how the agents navigate in more complex environments.

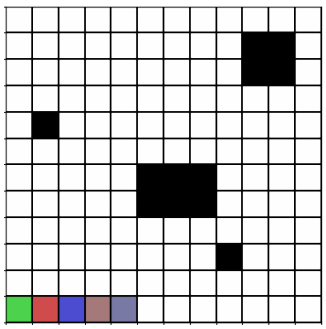

Figure 5(a) – 0 seconds

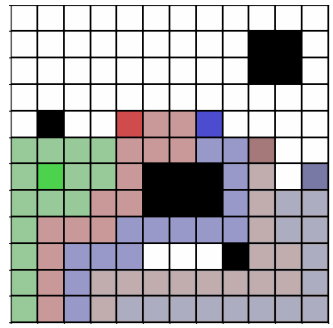

Figure 5(b) – 14 seconds

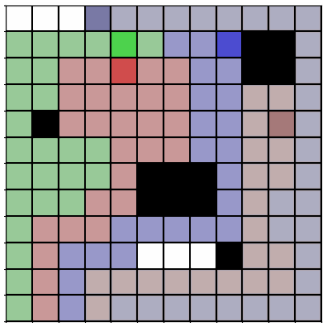

Figure 5(c) – 28 seconds

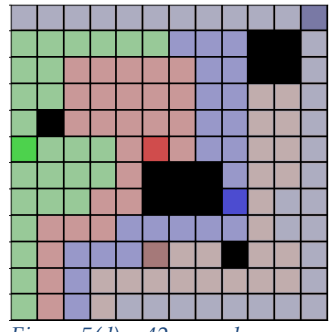

Figure 5(d) – 42 seconds

Figure 5 – This shows the trajectory of a Predator-Prey Algorithm over time with a swarm size of 5 in an environment size of 12 with obstacles. It shows the path of each agent around the obstacles.

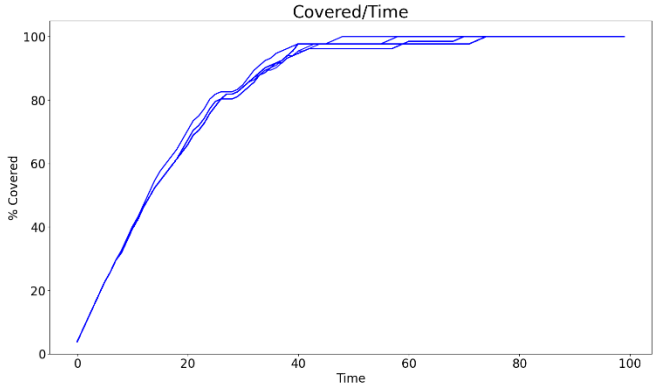

Figure 6 - Coverage/Time of a Predator-Prey Algorithm with swarm size 5 on an environment size 12 with obstacles. It helps to show consistency as well as that coverage is always achieved.

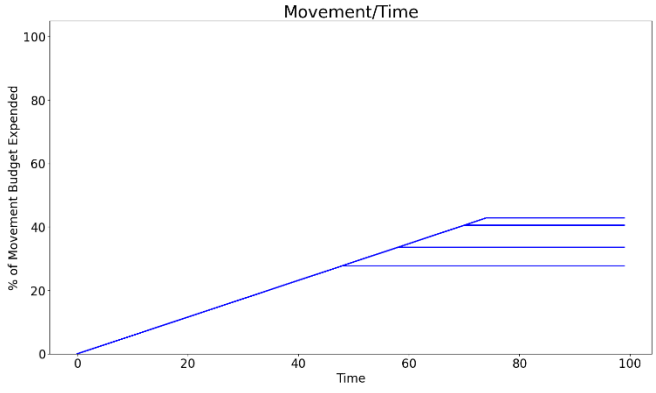

Figure 7 - Movement/Time of a Predator-Prey Algorithm with swarm size 5 on an environment size 12 with obstacles. This shows that efficiency is maintained even with obstacles.

The next plots show coverages in an environment of size 12 with differing swarm sizes, 3, 5, and 10 agents all using predator-prey learning. Obstacles were included in this environment. The same obstacle layout was used for each of the swarms to ensure the test was fair.

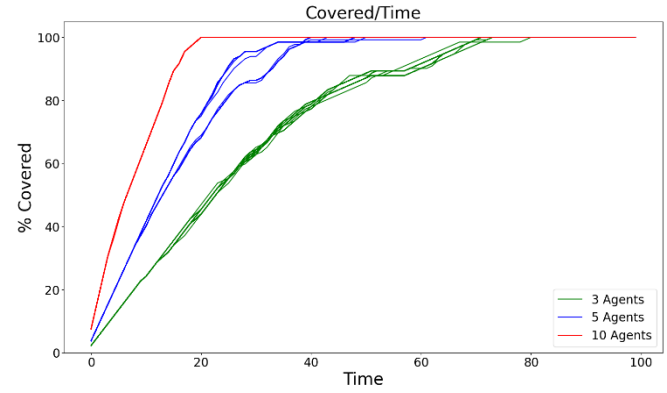

Figure 8 - The comparison of Coverage/Time different swarm sizes on an environment of size 12 with obstacles. Here you can see the difference in speed between the swarms.

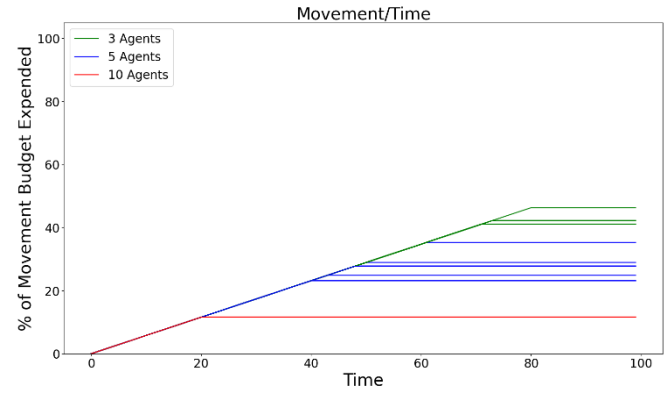

Figure 9 - The comparison of Movement/Time of different swarm sizes in an environment of size 12 with obstacles. This shows the efficiency of individual agents with different swarm sizes.

The following graphs demonstrate the comparison between the standard Q-learning and the version created in this paper. They were both run in the same environment, with a size of 12 with obstacles. Each algorithm was run for 50 loops and then the results of this simulation were plotted so they could be compared. A swarm of 5 agents was used for both algorithms to ensure a fair comparison.

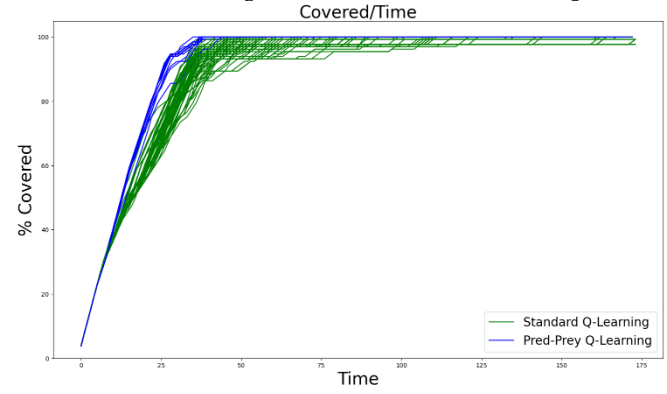

Figure 10 – Comparison between Predator-Prey and Standard algorithms Coverage/Time with obstacles. The large number of

lines shows the variation in the standard algorithm.

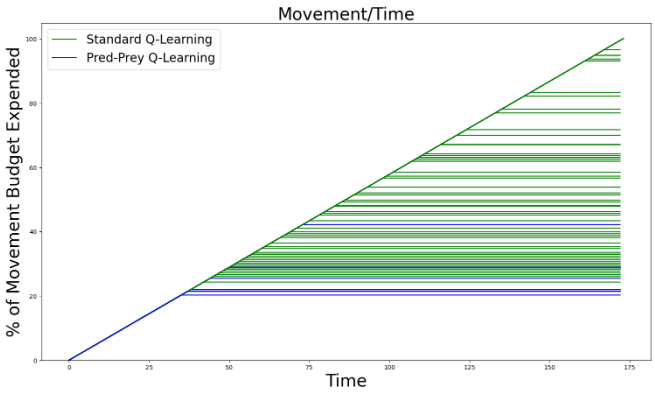

Figure 11 - Comparison between Predator-Prey and Standard algorithms Movement/Time with obstacles. The large number of lines shows the inconsistency of the standard algorithm.

5 Evaluation

The simulations show very promising results for the predator-prey swarm algorithm, when compared against the standard swarm algorithm it performs with greater consistency and speed.

In Figure 2 and Figure 5, the trajectory of the agents can be seen. It demonstrates how each agent explores their section of the environment; it also demonstrates the predator-prey methodology with the agents moving apart to explore different edges of the environment. The snapshots also show how the agents like to move in straight lines as well as how backtracking can be used to find unexplored locations missed by the agent.

Figures 3, 4, 6 and 7 show that even when obstacles are added the algorithm maintains its consistency with tight groupings of the simulations. It also shows that even with obstacles the algorithm always ensures that the environment is covered.

Figures 8 and 9 show the comparisons of different swarm sizes. This provides evidence of the effectiveness of the swarm as by adding additional agents the speed of the solution is drastically improved. These plots show that reliable results can be provided by any size swarm however the larger the swarm the better the results. This is useful to know as depending on the use case there may be different restrictions on the number of agents a swarm can have.

It can be seen in Figure 10 and Figure 11 that the predator-prey algorithm performs very well ensuring that the environment is always covered. Whereas with the standard Q-Learning, there are some cases in which coverage fails and the entire movement budget is expended. This can be seen in Figure 10, not every line (simulation) reaches 100% before the end of the time. This was an important criterion to meet for the solution ensuring that the environment can always be completely covered in a small number of steps, ensuring that the algorithm was efficient. This could potentially be further improved by improving the backtracking part of the algorithm.

Figure 10 and Figure 11 also show the improved algorithm is much more consistent providing similar results for all simulation loops, whereas the standard algorithm results vary drastically in its results. This was also important to ensure that the swarm would act predictably so that the user could know how the swarm would perform before using it, this is a very important factor in some potential use cases.

These results can be used to answer the research question "Can you adapt Q-learning to be more effective in Coverage Path Planning problems?" The results show that the adapted method is an improvement over normal Q-learning and therefore the paper has been successful in its aim to provide an improved Q-learning method. Returning to the aims of the paper set out in the introduction, they have all been successfully met. The resulting method ensures a coverage is always found, improves the efficiency of found coverages and is also much more reliable than the standard method.

6 Conclusions

This paper successfully provides an alternative method for the coverage path planning problem. It utilizes a swarm with agents using an adapted Q-learning algorithm to learn about their environment and plan an efficient path to explore it. The paper builds upon the existing Q-learning algorithm augmenting it with an adapted predator-prey method. The solution managed to successfully augment this method to allow agents of the swarm to simulate both prey and predator, this led to a more effective coverage as agents aimed to stay apart and therefore covered different sections of the environment. The adapted solution also performed much better in environments with obstacles which was another aim of this paper. The solution was much more efficient than a standard Q-learning swarm, ensuring that the environment was covered in fewer steps. Most importantly the improved method ensured that the environment was always fully covered when possible whereas the standard swarm would sometimes get stuck and be unable to complete a full coverage.

This adapted method has some particularly useful applications where efficient coverage is necessary, especially if the environment is irregular and contains obstacles. However, it does not consider 3D terrain and is therefore currently limited to ground applications such as lawnmowers, automatic vacuum cleaners or rover exploration.

The method can be built upon to further improve its efficiency, such as implementing a more efficient backtracking method. It could also be further adapted to more specialized uses, such as considering 3D to allow for use in UAVs. A dynamic environment could also be considered in which obstacles move, requiring the agents to adapt during coverage.

References

[1] Cao, Z.L., Huang, Y. and Hall, E.L., 1988. Region filling operations with random obstacle avoidance for mobile robots. *Journal of Robotic systems*, 5(2), pp.87-102.

[2] Cho, S.W., Park, H.J., Lee, H., Shim, D.H. and Kim, S.Y., 2021.

Coverage path planning for multiple unmanned aerial vehicles in maritime search and rescue operations. *Computers & Industrial Engineering*, 161, p.107612.

[3] Lin, H.Y. and Huang, Y.C., 2021. Collaborative complete coverage and path planning for multi-robot exploration. *Sensors*, 21(11), p.3709.

[4] Bouman, A., Ott, J., Kim, S.K., Chen, K., Kochenderfer, M.J., Lopez, B., Agha-mohammadi, A.A. and Burdick, J., 2022, October. *Adaptive coverage path planning for efficient exploration of unknown environments. In 2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)* (pp. 11916-11923). IEEE.

[5] Watkins, C.J.C.H., 1989. Learning from delayed rewards.

[6] Clifton, J. and Laber, E., 2020. *Q-learning: Theory and applications. Annual Review of Statistics and Its Application*, 7, pp.279-301.

[7] Navarro, I. and Matía, F., 2013. An introduction to swarm robotics. *Isrn robotics*, 2013, pp.1-10.

[8] Viet, H.H., Dang, V.H., Laskar, M.N.U. and Chung, T., 2013. BA*: an online complete coverage algorithm for cleaning robots. *Applied intelligence*, 39, pp.217-235.

[9] Prabakaran, V., Elara, M.R., Pathmakumar, T. and Nansai, S., 2018. Floor cleaning robot with reconfigurable mechanism. *Automation in Construction*, 91, pp.155-165.

[10] Hameed, I.A., 2017, July. Coverage path planning software for autonomous robotic lawn mower using dubins' curve. In *2017 IEEE International Conference on Real-time Computing and Robotics (RCAR)* (pp. 517-522). IEEE.

[11] Höffmann, M., Clemens, J., Stronzek-Pfeifer, D., Simonelli, R., Serov, A., Schettino, S., Runge, M., Schill, K. and Büskens, C., 2022, December. Coverage path planning and precise localization for autonomous lawn mowers. In *2022 Sixth IEEE International Conference on Robotic Computing (IRC)* (pp. 238-242). IEEE.

[12] Huang, K.C., Lian, F.L., Chen, C.T., Wu, C.H. and Chen, C.C., 2021. A novel solution with rapid Voronoi-based coverage path planning in irregular environment for robotic mowing systems. *International Journal of Intelligent Robotics and Applications*, 5(4), pp.558-575.

[13] Hameed, I.A., 2014. Intelligent coverage path planning for agricultural robots and autonomous machines on three-dimensional terrain. *Journal of Intelligent & Robotic Systems, 74(3)*, pp.965-983.

[14] Wang, L., Wang, Z., Liu, M., Ying, Z., Xu, N. and Meng, Q., 2022. Full coverage path planning methods of harvesting robot with multi-objective constraints. *Journal of Intelligent & Robotic Systems*, 106(1), p.17.

[15] Dahiya, C. and Sangwan, S., 2018. Literature review on travelling salesman problem. *International Journal of Research*, 5(16), pp.1152-1155.

[16] Arkin, E.M., Fekete, S.P. and Mitchell, J.S., 2000. Approximation algorithms for lawn mowing and milling. *Computational Geometry*, 17(1-2), pp.25-50.

[17] Galceran, E. and Carreras, M., 2013. A survey on coverage path planning for robotics. *Robotics and Autonomous systems*, 61(12), pp.1258-1276.

[18] Tan, C.S., Mohd-Mokhtar, R. and Arshad, M.R., 2021. A comprehensive review of coverage path planning in robotics using classical and heuristic algorithms. *IEEE Access*, 9, pp.119310-119342.

[19] Xing, B., Wang, X., Yang, L., Liu, Z. and Wu, Q., 2023. An algorithm of complete coverage path planning for unmanned surface vehicle based on reinforcement learning. *Journal of Marine Science and Engineering, 11(3),* p.645.

[20] Jia, Y., Zhou, S., Zeng, Q., Li, C., Chen, D., Zhang, K., Liu, L. and Chen, Z., 2022. The UAV path coverage algorithm based on the greedy strategy and ant colony optimization. *Electronics, 11(17),* p.2667.

[21] Nasirian, B., Mehrandezh, M. and Janabi-Sharifi, F., 2021. Efficient coverage path planning for mobile disinfecting robots using graph-based representation of environment. *Frontiers in Robotics and AI, 8*, p.624333.

[22] Zhou, B., Zhang, Y., Chen, X. and Shen, S., 2021. Fuel: Fast uav exploration using incremental frontier structure and hierarchical planning. *IEEE Robotics and Automation Letters, 6(2),* pp.779-786.

[23] Sutton, R.S. and Barto, A.G., 2018. *Reinforcement learning: An introduction*. MIT press.

[24] Piardi, L., Lima, J., Pereira, A.I. and Costa, P., 2019, July. Coverage path planning optimization based on Q-learning algorithm. In *AIP Conference Proceedings* (Vol. 2116, No. 1). AIP Publishing.

[25] Zhou, Q., Lian, Y., Wu, J., Zhu, M., Wang, H. and Cao, J., 2024. An optimized Q-Learning algorithm for mobile robot local path planning. *Knowledge-Based Systems*, 286, p.111400.

[26] Puente-Castro, A., Rivero, D., Pazos, A. and Fernandez-Blanco, E., 2022. UAV swarm path planning with reinforcement learning for field prospecting. *Applied Intelligence*, 52(12), pp.14101-14118.

[27] Zhang, M., Cai, W. and Pang, L., 2023. Predator-prey reward based Q-learning coverage path planning for mobile robot. *IEEE Access*, 11, pp.29673-29683.