Intelligent Garbage Collection Agent: A Comparison of A* Search and Reinforcement Learning

Clarine Tan Kaili
School of Computer Science
University of Nottingham
Nottingham, UK
hfyct2@nottingham.ac.uk

I. Introduction

With the rise of waste around the world, the need for an effective garbage collection system increases. Waste is typically collected regularly with a fixed schedule, and it can be challenging to determine the most effective schedule that also minimises the costs. To improve waste collection systems, intelligent agent-based systems have been proposed as a solution. This paper compares the performance of two different algorithms, A* and reinforcement learning, in developing an intelligent garbage collecting agent. This study also evaluates the effect of different starting points for the agent on the performance of the A* and reinforcement learning algorithm, specifically Q-learning. By varying the agent's starting points using four different coordinate pairs, the robustness and adaptability of both algorithms in different scenarios can be assessed. The result of this study has highlighted the importance of hyperparameter tuning in reinforcement learning. Furthermore, this paper discusses the limitation of the study conducted and several future research directions. Additionally, this paper discusses the practicality of these intelligent garbage collection methods in real-life scenarios.

II. Literature Review

Waste generation has been on the rising for several years due to reasons such as economic growth and population growth. This raises the concern for maintaining cleanliness and good hygiene in cities. Real-world examples from the UK were discussed by [1], demonstrating the impact of vehicle routing and scheduling on waste management. The use of AI techniques has been proposed to improve the efficiency of garbage collection systems.

A* search algorithm is a popular technique that is commonly used to solve pathfinding problems. The garbage collection system is a type of vehicle routing problem (VRP) that can be solved with the A* search algorithm. Heuristics methods had shown promising results for solving VRP when tested against 486 benchmark tests [2].

Reinforcement learning has been a popular solution in solving VRP. A study has shown promising results in solving VRP using reinforcement learning techniques [3]. It was found that it outperforms classical heuristic methods [3]. However, the solution found in [3] was shown to be impractical in real-world scenarios due to its extremely long training time. Another study was done highlighting this problem and suggests an alternative to overcome the curse of dimensionality which is the cause of the high training time using the model-free RL SolVeR Pro [4].

III. Methodology

A. Environment and Agent Designs

Fig. 1 shows the environment that is used in this study simulates a garbage collection system. It consists of a 7 by 7 grid. The white grids represent the paths the agents can navigate, and the grey grids represent walls that serve as obstacles. The grey grids are intended to represent roads in a real-world setting. The brown grid at (1,3) represents the landfill where the agent can empty its load.

There are 4 bins in the environment as shown in Fig. 1, each assigned to a specific area that is identifiable by its colour. 2 red bins are located in 'busy' areas and fill up at a rate of 8 units every 5 moves, while the orange-coloured bin represents bins in 'moderately busy' areas and fills up at a rate of 5 units every 5 moves. The yellow bin is in 'quiet' areas and fills up at a rate of 3 units every 5 moves. Each bin is labelled with a number indicating its current trash level, with a maximum load capacity of 50 units. The bins will continue to fill up even when the agent is collecting from them.

The blue triangle in Fig. 1 represents the agent with its current load level labelled on it. The agent can hold up to 100 units of trash and can move up to 150 steps. The agents can only collect or empty loads when it is directly on the grids with a bin and the landfill grid. To ensure a fair comparison, the agent will have 4 different starting positions for each run of both algorithms. However, for the second algorithm, the same 4 starting positions will be used in each run to maintain consistency.

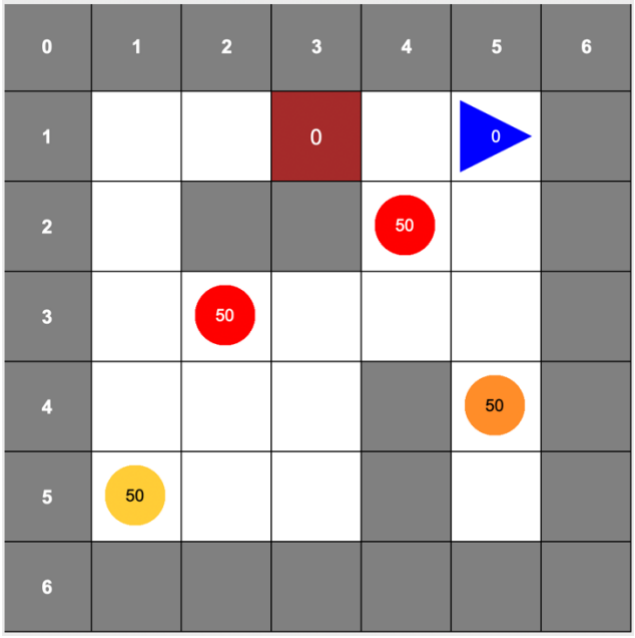

Fig. 1. The environment of the experiment

B. Technologies used

1) Programming language and libraries

This project is implemented using the Python programming language and several libraries. Heapq is used to implement the priority queue for the A* algorithm. The NumPy library is mainly used for creating and manipulating the Q-table for reinforcement learning which stores the expected reward for each action in all possible states. Numpy also includes random functions which are used for selecting random actions during the reinforcement training process. To visualise how the agent interacts with the environment, the Tkinter library is used to create the graphical user interface (GUI). Lastly, the Time library is used to control the speed of resetting the experiment runs. It ensures that there is a delay between runs for a text to be displayed in the GUI indicating that the experiment has ended, and the environment will be reset.

2) A algorithm*

The A* algorithm is a well-known best-first search algorithm used for pathfinding. It maintains a priority queue of nodes to explore. In this study, the nodes represent the grids in the environment. At each iteration, the algorithm explores the node with the lowest estimated cost which is determined by the number of steps taken to reach that node and the predicted number of steps needed to reach the end point. A heuristic function is needed to predict the cost needed to reach the end point. In this study, the Euclidean distance is used as the heuristic function because it is admissible. As a result, the algorithm is optimal. However, the A* algorithm only finds the shortest path from a start and an end point. Fig. 2 shows how the A* algorithm was implemented to travel to multiple locations.

Algorithm 1 A* Search Algorithm

```
uncollected bins ← all bins
while agent remain move > 0
    if agent load < max agent load capacity
        find nearest uncollected bin using A* algorithm
        go to nearest bin and collect trash
        remove the bin from the list of uncollected bins
    else
        find best path to landfill
        uncollected bins ← all bins
```

Fig. 2. Pseudocode of A* algorithm implementation

3) Reinforcement Learning with Q-Learning

Q-learning is a model-free reinforcement learning algorithm that enables an agent to be trained in a dynamic environment, which is typical of real-life scenarios. To train the agent using reinforcement learning, the state and actions need to be defined. In this study, the state of the agent is defined as the agent's location and its garbage load. The actions that the agents can take are moving up, down, left and right. When the agent reaches a bin location, it will collect the garbage and empty its load at the landfill.

The goal of Q-learning is to find the optimal policy which is a mapping of states to actions that maximises the expected cumulative reward over time. The agent has a "memory" in the form of a Q-table. The Q-table stores the expected cumulative reward (Q-value) for each action in each state. The Q-values were initialised to 0 at the beginning of the program. The Q-value is updated over time as the agent explores the environment.

During the training process, the agent selects its next action using the epsilon-greedy approach. This allows the agent to explore new actions based on the epsilon value. In this study, the agent randomly explores a new state 10% of the time in hopes of discovering potentially better actions. When the agent takes an action, it observes the resulting reward and next states that is stored in the Q-table. The Q-values are updated using the Bellman equation, as shown in equation (1).

$$Q(s,a) = Q(s,a) + lr + (r + df * max(Q(s',a'))) \quad (1)$$

Where:

- Q(s,a) is the Q-value for taking action a in state s
- lr is the learning rate
- df is the discount factor
- a' is the action with the highest Q-value in the next state
- s' represents the next state upon taking action a'
- r is the reward received from taking action a in state s which is calculated based on Table I.

The optimal policy will be found when the agent is trained till the Q-values converges. However, this is not the case for our study as seen in Fig. 3. The agent is not being trained until the Q-values converge, but instead it is determined by a time lapse. The time lapse is three times the maximum number of steps that the agent can take. This is a trade-off between reducing the training time and maintaining its performance.

Algorithm 2 Reinforcement Learning with Q-Learning Policy

```
initialise states, actions, q-table
initialise hyperparameters (learning rate, discount factor, epsilon)
while episode in range(1000)
    initialise environment and current state
    while time lapse is not 0
        choose action for current state
        determine the reward and next state
        update the q value for current state and action
        current state ← next state
```

Fig. 3. Pseudocode of reinforcement learning with Q-learning policy implementation

TABLE I. REWARD FOR EACH STATE AND ACTION

State/Action	Given Reward
Move a step	+10
Collect trash from a bin (agent is on a grid with bin)	*Less than 20 units of trash collected:* + units of trash collected * 2
	At least 20 units of trash collected: + units of trash collected * 3
Empty trash at the landfill (agent is on the landfill grid)	*Less than 40 units of trash emptied:* + units of trash emptied * 1.5
	40 to 69 units of trash emptied: + units of trash emptied * 3
	Above 69 units of trash emptied: + units of trash emptied * 4
Collision (agent is on a wall/obstacle grid)	-1500
Uncollected trash from a bin (bin is not empty)	*For each bin:* - (40 + units of trash in the bin)

4) Implementation challenges

Initially, the plan was to use an existing third-party environment from the OpenAI gym. But while attempting to modify the environment, it was found that the process was more complex than anticipated due to the lack of understanding of the program. As a result, a new environment was created using the Tkinter library to create a suitable simulation for the project.

Besides the A* algorithm and reinforcement learning, Dijkstra's algorithm was also considered. However, it was found to be unsuitable for this project as it does not use any heuristic which makes it computationally expensive. In the experiment, the shortest path to the nearest bin needs to be re-computed upon reaching a bin or landfill to determine the next course of action for both A* and Dijkstra's algorithms. Since the A* algorithm is a modification of Dijkstra's algorithm that uses a heuristic function, it was determined that implementing Dijkstra's algorithm would yield similar performance to the A* algorithm in terms of the total amount of trash collected but with a much longer computation time.

The most challenging part of this project was to tune the hyperparameters for reinforcement learning. In addition to the learning rate, discount factor, and epsilon, the reward function must also be tuned to optimise the agent's performance. However, due to the high number of hyperparameters and possible combinations, it was difficult to determine the best hyperparameter. Despite numerous attempts, the best set of hyperparameters was not found, which is reflected in the agent's performance during experiment runs. Moreover, the experiment has two goals, collecting trash from bins and emptying it at the landfill. Balancing these goals was challenging as it was important to ensure that the agent prioritise collecting from bins while it is relatively empty and focused on emptying its load at the landfill when it is near full capacity. This was a significant challenge in the project.

IV. EXPERIMENTS AND RESULTS

A. Experiment design

To evaluate the performance of the proposed algorithms, two experiments were conducted. The first experiment used the A* algorithm, while the second experiment used reinforcement learning. Each experiment was repeated 4 times to investigate the effect of changing the agent's starting point. Specifically, the starting point of the agent was varied using four different coordinate pairs: [(1,1), (3,3), (5,5), (1,5)].

Reinforcement learning is a stochastic algorithm which means that its performance can vary in each trial. Thus, each experiment using reinforcement learning was repeated three times to obtain an average performance measure. On the other hand, the A* algorithm uses an admissible heuristic function, which guarantees that it will always find the optimal path for a given starting point. Although there is no need to run multiple trials with the same starting point as the algorithm's performance will be consistent, it was done to ensure consistency and to check that the algorithm is performing as intended.

B. Results

Fig. 4 shows the average total unit of garbage collected, hereafter referred to as the score, across 3 trials for the 4 different starting positions. Overall, the A* algorithm performed better than the reinforcement learning models for all the different starting positions, with the highest score of 856 at starting position (5,5). The performance of the 3 trials for reinforcement learning is shown in Fig. 5, with trial 2 achieving the best overall score with no 0 scores recorded for any starting position. No clear trend was observed in the performance of the reinforcement learning models as at least one trial has scored 0 in 3 out of the 4 trials.

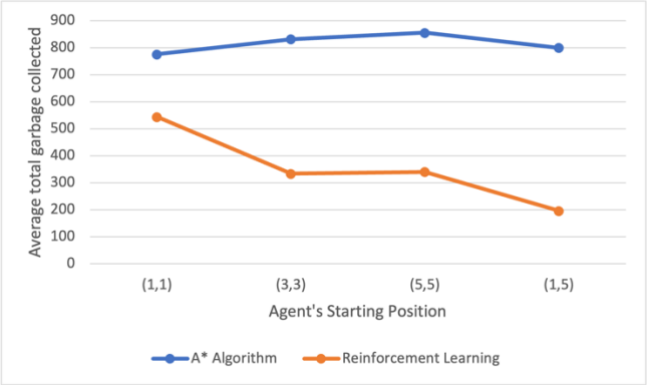

Fig. 4. The average total garbage collected across 3 runs of the 4 different agent's starting positions

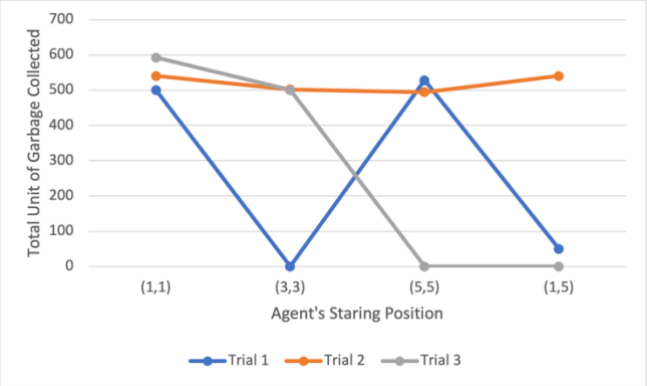

Fig. 5. The total garbage collected across 3 trials of the 4 different agent's starting positions when the agent is trained using reinforcement learning

V. DISCUSSION

The results of the experiments show that the A* algorithm outperforms the reinforcement learning models for all the different starting positions. This is likely due to the ability of the algorithm to search for the optimal path systematically while avoiding obstacles. In contrast, the reinforcement

learning models rely on trial and error to learn the optimal policy, which may be less efficient.

However, this does not necessarily imply that the A* algorithm is superior to reinforcement learning as the hyperparameters of the reinforcement learning models are not tuned well. As the performance of the reinforcement learning models is not stable, it is difficult to make a direct comparison between the two algorithms. This is supported by the observation in Fig. 5 as there is a fluctuation in the performance with no clear trend across the 3 trials for the same starting position.

Despite that, the experiment shows that the A* algorithm is stable and reliable as it consistently achieved similar high scores across all the different starting positions. This indicates that the algorithm is robust to different starting conditions which is important in real-world scenarios where the environment conditions may vary. Thus, using A* algorithm to implement garbage collection systems will result in a reliable system with consistent performance.

On the other hand, the reinforcement learning models achieved both decent and bad scores for the same starting position in different trials since the algorithm is stochastic. This shows that the model does not generalise well as it is likely to overfit to trial 2. Moreover, this indicates that reinforcement learning models are sensitive to hyperparameters and must be carefully tuned to achieve good performance.

Trial 2 achieved the best overall score across 3 trials conducted using reinforcement learning. But it is important to note that this may be due to overfitting to that specific trial. In an attempt to improve the model's performance while adjusting the hyperparameters, using the same random seed was necessary to ensure reproducibility. This was needed to observe the effect of hyperparameter changes. The findings from trial 2 suggest that the reinforcement learning models have the potential to perform well with further parameter tuning.

It is worth noting that the reinforcement learning models that obtained a score of 0 show that the agent was not able to collect any garbage. Other than poor hyperparameter tuning, it is likely that this is also caused by the low number of episodes or time steps. As mentioned previously, the time steps were determined by the maximum number of steps that the agent can take. This indicates that the Q-values were not able to converge within the limited number of time steps, resulting in a poorly learned policy. This suggests that reducing the training time has a severe impact on the model performance. Therefore, this highlights the critical role of adequate training time and emphasises that it should not be compromised especially in real-world scenarios where the environment is much more complex. It is important to carefully consider the training time and ensure that the model performance is not compromised too much.

Despite the efforts made in this study, the results obtained were not very promising, especially for reinforcement learning models which may be due to the limitations of this study. These limitations need to be considered when interpreting the findings. Firstly, the study only focuses on a single environment. The bins are located at fixed positions and the obstacles were unchanged throughout the experiments. As such, the findings of this paper may be limited to the context of our study.

Another limitation of this study that needs to be highlighted is that the simulated environment does not accurately reflect the complexity of actual environments. Factors such as traffic and weather conditions were not considered in our study. In real-world situations, the distributions and number of bins are more varied. Therefore, the results of this study should not be directly applied to real-life situations. Future studies should investigate the performance of the A* algorithm and reinforcement learning, specifically Q-learning, in much varied and complex environments, including unpredictable situations such as traffic.

Besides that, future studies could also investigate the potential of other algorithms, such as ant-colony optimisation and genetic algorithms, for implementing effective garbage collecting systems. In addition, as the study has found difficulty in implementing and tuning the reward function for reinforcement learning, it may be beneficial to explore alternative reward functions. Investigating the impact of the hyperparameter choice on the model performance can provide insights into designing effective garbage collection systems.

VI. Conclusion

While the study did not provide a definite comparison between the A* algorithm and Q-learning to optimise garbage collecting systems, it shed light on the challenges of hyperparameter tuning in reinforcement learning. The difficulties in tuning the reward function resulted in poor generalisation and poor overall performance of the reinforcement learning models. Therefore, future research directions should address these challenges in addition to the limitations of this study. Additionally, the performance of other algorithms should be done to determine the most suitable algorithm to implement effective garbage collecting systems. In conclusion, this study shows that using the A* algorithm to implement a garbage collecting system will lead to a stable and reliable system while future research should be done to determine the practicality of implementing garbage collecting systems with q-learning. This study also highlights the importance of hyperparameter tuning in the development of effective garbage collection systems.

References

[1] F. McLeod and T. Cherrett, "Waste Collection," in *Waste: A Handbook for Management*, 2011. doi: 10.1016/b978-0-12-381475-3.10004-x.

[2] P. M. Pardalos and S. Ropke, "A general heuristic for vehicle routing problems," *Computers & Operations Research*, vol. 34, no. 8, pp. 2403–2435, Aug. 2007, doi: 10.1016/j.cor.2005.09.012.

[3] M. Nazari, A. Oroojlooy, M. Takáč, and L. H. Snyder, *Reinforcement learning for solving the vehicle routing problem*, vol. 31. 2018, pp. 9861–9871. [Online]. Available: https://papers.nips.cc/paper/8190-reinforcement-learning-for-solving-the-vehicle-routing-problem.pdf

[4] A. K. Kalakanti, S. Verma, T. Paul, and T. Yoshida, *RL SolVeR Pro: Reinforcement Learning for Solving Vehicle Routing Problem*. 2019. doi: 10.1109/aidas47888.2019.8970890.