Received 16 January 2025, accepted 14 March 2025, date of publication 19 March 2025, date of current version 31 March 2025.

Digital Object Identifier 10.1109/ACCESS.2025.3553036

RESEARCH ARTICLE

Reinforcement Learning for Optimize Coverage in Art Gallery Problem Using Q-Learning Based in Grid World

YUAN-HSUN LIAO[1], **PO-CHUN CHANG**[1], **AND HSIAO-HUI LI**[2]
[1]Department of Computer Science, Tunghai University, Taichung 407224, Taiwan
[2]Department of Maritime Information and Technology, National Kaohsiung University of Science and Technology, Cijin Campus, Kaohsiung 805301, Taiwan

Corresponding author: Hsiao-Hui Li (Xiasohui@gmail.com)

ABSTRACT This paper presents a novel approach to solving the Art Gallery Problem (AGP) using a grid-based system and Reinforcement Learning (RL) in a two-dimensional space. The algorithm converts complex polygons into grids to simplify coverage calculations, allowing for scalable extensions. The method employs a Q-learning agent that interacts with the environment to optimize guard placement by balancing the number of guards and the coverage area. Experiments show that finer grid densities improve accuracy, while parameter adjustments (α, γ, ε) significantly impact performance. The algorithm effectively handles varying map complexities, demonstrating robust and efficient coverage solutions.

INDEX TERMS Reinforcement learning, art gallery problem, Q-learning.

I. INTRODUCTION

In the Art Gallery Problem (AGP) [1], [2], [3], the algorithm emphasizes minimizing the number of guards required. The AGP considers three main issues: minimizing the number of guards, maximizing the coverage area, and enhancing the patrol capability [2], [3]. Art gallery problems display an important role in enhancing the understanding of visibility and coverage within polygonal spaces.

The past method for solving art gallery problems depends on mathematical formulation [3], greedy heuristics [4], [5], and integral linear programming [4] to solve art gallery problems. There are two main problems with these methods. The mathematical formulation-based method has an explained ability in coding, directly adding development cost to adjust detail design. Also, the mathematical formulation-based method less the escape ability to find a globally optimized solution. The determine algorithm, like greedy heuristic [4], [5] and integral linear programming [4], is weak in calculating times. Along with the size of the map increases, the determined algorithm would spend too much time to calculate. Moreover, these types of algorithms could not stabilize the globally optimized solution. Experiment 5 proves these properties: as the map gets bigger, the determined algorithm would spend too much time to calculate, and the solution would not be the globally optimized solution.

In summary, this study uses a grid-based method and Q-learning to solve the Art Gallery Problem [1], [2] and has achieved good results in experiments. Experiments 1 and 2 test the combination of algorithm parameters. Experiment 3 tests the performance of reinforcement learning at different scale maps to solve an art gallery; the result is reinforcement learning can process different maps and achieve a positive result. Experiment 4 tests the reinforcement learning performance in a real-world case; the result is reinforcement learning can be applied in real-world cases with good results. Experiment 5 tests the performance between reinforcement learning, greedy heuristic [4], and integral linear programming [4]. At the small map, reinforcement learning gets the minimum average number of guards. At the large map, reinforcement learning gets the minimum average number of guards and running times. In conclusion, compared to greedy heuristics [4] and integral linear programming [4], reinforcement learning is more suitable as a method to solve art gallery problems.

The associate editor coordinating the review of this manuscript and approving it for publication was Anandakumar Haldorai.

II. LITERATURE REVIEW AND BACKGROUND

To understand the approach of this paper, it is essential to review the classical theories from previous foundational studies, above view optimize technique [6], art gallery problem [1], [2], greedy heuristic [4], Q-learning algorithm [17], polygon attribute [20], and reinforcement learning [8].

A. VIEW OPTIMIZE AND SELECTION PROBLEM

The view optimization and selection problem has been extensively studied and generalized in previous works [6], [12]. This paper [12] mentions the main trend of this issue: one is materializing views in a database to speed up query processing, and another is selecting views to materialize in a data warehouse to answer decision support queries. In this paper, the algorithm takes our viewpoint into the database. In the view optimization problem, the algorithm will search the different combination views into one map. To enhance generalization capability, the algorithm uses a database to facilitate the optimization process [6]. Also, in the view optimization and selection problem, the algorithm considers reducing the evaluation costs of the queries [6]. Moreover, the algorithm needs to consider the materialized view [7]. In summary, to solve and design view optimization and selection problems, one needs to consider much detail. At the same time, this problem is a popular issue to reach.

B. ART GALLERY PROBLEM

The art gallery problem involves figuring out the least number of edge guards and point guards needed to cover a simply connected polygon area, and both of these tasks are NP-hard [1]. As a classical computation geometry problem, have been reached in long history [2], [3]. Many Reacher properties are novel approaches to solving art gallery problems, iterative primal-dual relaxation [13], greedy heuristics [4], and graph theory [14].

C. GREEDY-HEURISTICS METHOD

The greedy heuristics method is a framework to get the current solution to each different problem. This framework has been generalized applicate into lots of development. For example, this paper [15] used a greedy selection solution to solve ordering problems. Urszula Stańczyk and Beata Zielosko proposed a heuristic-based selection method to solve the rough set problem [5]. Heuristic-based enhances the performance of selects and reduces relevant features and cardinalities in data sets. Akcay et al. offer a greedy-based algorithm to solve multidimensional knapsack problems [16]. In summary, the greedy heuristics method is a powerful framework to fit a different tough task.

D. Q-LEARNING ALGORITHM

The Q-learning algorithm is arguably one of the most applied representative reinforcement learning approaches and one of the off-policy strategies [17]. Q-learning algorithm has been used in reinforcement optimization learning and artificial intelligence problems. Q-learning is a reinforcement learning algorithm that involves exploring the environment randomly and updating actions dynamically based on the received rewards. Q-learning has a huge potential in different tasks, likewise, continuing sensor space detection [18] and hardware architecture [19].

E. POLYGON ATTRIBUTE

The perimeter is the sum of the lengths of all sides of a polygon [20]. The n is the number of vertices of the polygon. v_i and v_{i+1} are adjacent vertices, distance(v_i, v_{i+1}) is the distance between these two vertices.

$$P = \sum_{i=1}^{n} \text{distance}(v_i, v_{i+1}) \tag{1}$$

For a vertex with coordinates (x, y), the area [21] formula is

$$A = \frac{1}{2} \left| \sum_{i=1}^{n} (x_i \cdot y_{i+1} - x_{i+1} \cdot y_i) \right| \tag{2}$$

The coordinates (x_{n+1}, y_{n+1}) are considered identical to (x_1, y_1), indicating that the polygon is closed.

Number of vertices is the number of vertices of the polygon.

$$n = \text{len}(vertices) \tag{3}$$

Compactness reflects how similar the shape is to a circle [22], with a value closer to 1 indicating that the shape is closer to a circle.

$$C = \frac{P^2}{4\pi A} \tag{4}$$

Here, the P is the perimeter and A is the area.

The edge length standard deviation reflects how regular the polygon is [23]. A larger standard deviation indicates greater differences in side lengths, making the shape more irregular. Conversely, a smaller standard deviation means the side lengths are more uniform, resulting in a more regular polygon.

$$\text{std} = \sqrt{\frac{1}{n-1} \sum_{i=1}^{n} (d_i -)^2} \tag{5}$$

n is the number of edges. d_i is the length of the i-th side.

The circumscribed radius is the distance from the center of gravity of the polygon to the furthest vertex of all the vertices [24]. For a polygon, the circumcircle is a circle containing all vertices, and the circumcircle radius is the radius of this circle.

$$R_{\text{circ}} = \max(\text{distance}(c, v_i)) \tag{6}$$

c is the center of gravity of the polygon. v_i is the *ith* vertex of the polygon.

Inscribed radius is the shortest distance from the polygon's center of gravity to its side [25]. For a polygon, its inscribed circle is a circle that is tangent to all sides of the polygon, and the radius of the inscribed circle is the radius of this circle. c is the center of gravity of the polygon. e_i is ith edge of the polygon.

$$R_{\text{ins}} = \min(\text{distance}(c, e_i)) \tag{7}$$

The radius of the inscribed circle represents the maximum size of the free inside the polygon [26]. For a given polygon, a large inscribed circle radius means that the polygon's shape is more symmetrical and the internal space is better utilized. While a smaller inscribed circle radius stands for the shape is irregular and some edges are closer together.

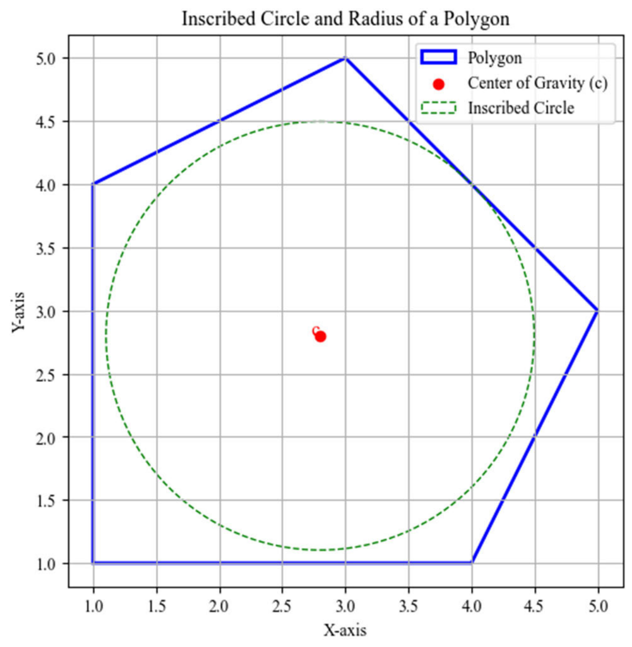

FIGURE 1. The diagram of the inscribed circle, blue boxed be polygon boxed, green circle is inscribed of center pointed, visualized as the round of gravity.

Figure 1 is the toy example to visualize the inscribed circle, the center of gravity (c), and the polygon. The inscribed circle is shown as a dashed green circle, and its radius (r_{ins}) is the shortest distance from the center of gravity to the polygon's edges.

F. REINFORCEMENT LEARNING

Reinforcement learning, as the popular learning model, is evaluated from dynamic programming [8]. Usually applicable to system control [9], playing game strategy [10], and simulation of animal learning [11].

Reinforcement learning commonly uses two strategies to evaluate the model [8]; one is to search in the space of behaviors in order to find one that performs well in the environment. The second is to use statistical techniques and dynamic methods to estimate the utility of making actions in the states of the world [27]. In this paper, the main algorithm is based on type 1 to emulate.

G. BACKGROUND KNOWLEDGE FOR ALGORITHM DESIGN

In this paper, the algorithm uses a grid-based method to convert a two-dimensional polygon into a grid system and consider the grid density. Grid density refers to the spacing between grids. Smaller spacing results in more grids and higher coverage accuracy [6], [7].

Next, the algorithm discusses the use of Reinforcement Learning (RL) [8], [9], [10]. Our method employs RL where the agent interacts with the environment to receive rewards and decide the next action [11]. The agent learns and adjusts its actions based on the interaction with the environment to achieve optimal coverage.

When defining the environment, the algorithm follows these steps: initializing the environment, designing the agent's actions, and defining the interaction process between the agent and the environment. Initialization involves setting the positions of points, all points in the map, and the action space. The reset function resets the agent's position and records the space at the beginning of each new iteration [8].

The agent's actions include determining the position of the guards, calculating the coverage area, and calculating rewards. Coverage calculation uses a nine-grid method and considers the influence of obstacles. Regarding rewards, the algorithm aims for fewer guards and larger coverage areas.

In loop design, the agent repeatedly interacts with the environment and learns. The algorithm uses the Q-learning method for reinforcement learning. In the methodology section, the algorithm describes the parameter settings, their meanings, and the update of hyperparameters in detail.

Our experiment design includes three parts: the impact of different densities on the map, parameter experiments (testing values of α, γ, and ε), and experiments on maps of varying complexity. The results show that smaller density leads to larger exploration space; α value has the most significant impact on a single agent, while ε value has the most significant impact when testing generalization capability; as the map becomes more complex, the algorithm still converges within a reasonable number of iterations, and the exploration space increases with complexity.

III. METHODS

The first step is importing the algorithm to transform the polygon into a grid. This paper uses the transform algorithm form this paper [4]. Called discretization algorithm.

The discretization algorithm needs to input the vertices V of a polygon. The expected output is the discretization $D\,(P)$ of polygon P. This algorithm has five steps. First, the algorithm computes the bounding of the polygon P, which is the smallest rectangle that entirely contains the polygon. Second, the algorithm determines the resolution, Δx and Δy, which are the grid spacing values in the x and y, respectively. Third, the algorithm created a regular grid of points within the bounding box BB using the specified resolution. Fourth, this algorithm filters the grid points to retain only those within the interior of the polygon P. Finally, the set of points $D(P)$

is updated to include the vertices V of the polygon P. This ensures that all original vertices of the polygon are part of the discretized representation.

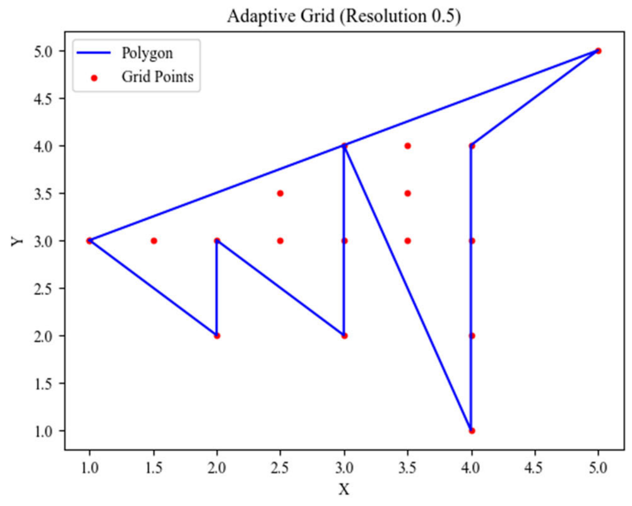

FIGURE 2. The grid with 0.5 value. The blue line boxed the polygon shape. The red points are extract by algorithm. 0.5 reflects the threshold of distance.

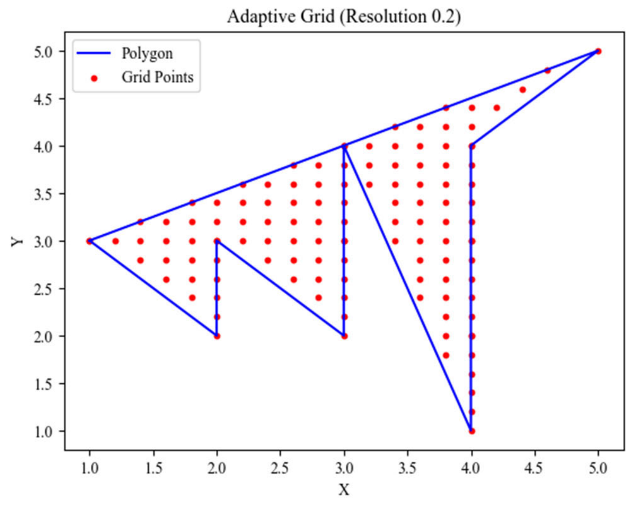

FIGURE 3. The grid with 0.2 value. The blue line boxed the polygon shape. The red points are extract by algorithm. 0.2 reflects the threshold of distance.

At the gym environment setup [28], first of all, the algorithm needs to decide the attributes and initialization. The points represent the positions of potential guard placements in the environment. These points are passed during the environment's initialization. Also, these points are loaded from this paper [4] algorithm.

The total wall is a list of indices representing the locations of obstacles. Guards cannot be placed at these points, and their field of view cannot pass through these points. Num of points represents the total number of points in the environment, derived from the number account of points. Current guard indices are a list that keeps track of the indices where guards are currently placed. This list starts empty and is updated as guards are added or removed. Action space is defined as the possible action in the environment. It is a two-way discrete space; 0 stands for removing a guard, and 1 stands for adding a guard to a guard.

Observation space represents the state space of the environment. It is defined as a discrete space with the number of points, indicating the number of guards currently placed.

Second, the algorithm needs to set up the reset method. The reset method initializes the environment for a new episode. It resets the current guard indices list to the empty and returns the initial observation, which is the number of guards placed.

Finally, the algorithm defines the step handling. It can be split into three parts: action handling, field of view calculation, and reward calculation. The action handling method takes an action and updates the environment's state. While choosing the action 0, meaning removing a guard, if there are guards present, it randomly removes one from current guard indices. Similarly, choose action 1, meaning that add a guard at a random index from the points. However, the method ensures that the guard is not added at a wall index by filtering current guard indices. Reward calculation is based on the number of unique points covered by the guards' fields of view. The more points covered, the higher the reward. If no guard has a field of view, the reward is zero.

Field-of-view calculation is a complex task. The algorithm defines this function as finding parallel and diagonal indices. First, the algorithm needs to identify the guard's position. The guard position is a list from the points array. Second, the algorithm needs to find parallel and diagonal indices. In this research, the algorithm defining the view of the guard is following this paper [29]. The visibility is two points u and v in a polygon P are said to be visible if the line segment joining u and v lies entirely inside P. Hence, the algorithm defines the visibility in the grid polygon in the six directions of grid points, which include parallel and diagonal. The algorithm split it into two parts, parallel points and diagonal points.

To find the parallel points, if the indices of all points where the x-coordinate equals the axis of the guard, this indicates vertical alignment with the guard. Else, if the indices of all points where the y-coordinate equals the axis of the guard, this indicates horizontal alignment with the guard. To find the diagonal points, first of all, the algorithm needs to understand the diagonal conditions. The diagonal condition identifies points where the absolute difference in x-coordinates equals the absolute difference in y-coordinates, implying the points lie on the same diagonal as the guard.

After that, the algorithm needs to categorize the diagonal points. Left-bottom points where x of this point is less than the x-axis of the guard and y of this point is less than the y-axis of the guard. Left-top is the point where x of this point is less than the axis of the guard and y of this point is bigger than the axis of the guard. Right-bottom is a point where the x of this point is bigger than the axis of the guard and the y of this point is less than the axis of the guard. Right-top points where x of this point is bigger than the x-axis of the guard and y of this point is bigger than the y-axis of the guard. After all, this part outputs a dictionary containing an array of indices for each alignment category.

While getting the alignment dictionary, the algorithm officially starts the calculation process; this functions the algorithm called calculate field of view. This method determines the set of points that are visible to the guard, taking into account the positions of walls that may block the guard's line of sight. The key is the logical determination of the wall. As the wall indices, the algorithm makes the filters and sorts a list of wall indices that are considered walls, based on whether their indices are present in another alignment dictionary key. On the other hand, include only those elements that are also in the alignment dictionary, and save this result as the new array.

Now, the algorithm going to run the iterative. First, the algorithm needs to categorize the walls. For each direction, walls are categorized on their indices relative to the guard's index. Smaller indices are the walls before the guard in the view direction. Larger indices are the walls after the guard in the view direction.

Second, the algorithm needs to determine the walls closest to the guard. In this process, the aim is to find the smallest index in larger indices and the largest index in small indices. If all the walls are after the guard, only the nearest one, which is the smallest index in larger indices, is considered. If all the walls are before the guard, only the nearest one, which is the largest index in small indices, is considered. Else if there are walls in both directions, include both closest walls for filtering.

After that, the algorithm filtered points based on the nearest Walls.

If only one wall is detected as the closest, filter the points of vertical parallel and horizontal parallel, based on this wall's position relative to the guard. The algorithm extracts the points, including points up to the wall and points from the wall onward.

If there are multiple walls in both directions, this algorithm filters points of vertical parallel and horizontal parallel between the closest walls. The points include between two sides, the guard and the closest wall before the guard, and the closest wall after the guard and other points.

In this situation, no matter if it is one wall or multiple walls. For each diagonal direction, apply similar logic to determine visible points by filtering. At the Left-top diagonal, the algorithm filter points to include only those beyond the furthest wall in the left-top direction. At the Right-Top diagonal, the algorithm filter points to include only those before the closest wall in the right-top direction. At the Left-Bottom diagonal, the algorithm filter points to include those beyond the furthest wall in the bottom-left direction. At the Right-Bottom diagonal, the algorithm filter points to include only those before the closest wall in the right-bottom direction.

Otherwise, if there isn't any wall, the view of each direction is the array from finding parallel and diagonal indices.

Fourth, the algorithm has to combine all of the views from each direction, including vertical, horizontal, and diagonal indices. As the reward calculation.

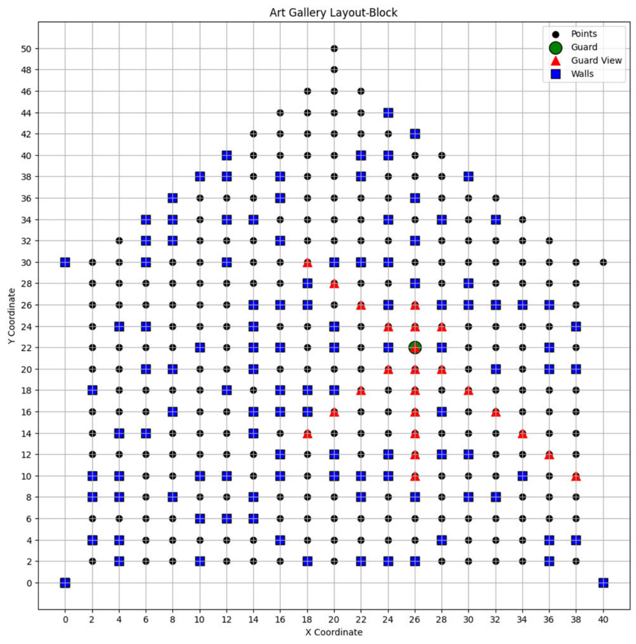

FIGURE 4. In the toy example of the guard view, the red grids are the guard's view, the green grids are the guard position, the blue grids are the critical wall position, and the black grids are the space.

In this paper, the algorithm chose the Q-Learning algorithm to optimize the art gallery problem. To achieve the Q-Learning implementation, the algorithm needs to definite Q-table, choose actions, how to update the Q-table, and the main training loop.

In the design of Q-table initialization, the Q-table is a matrix where each row represents a state and each column. The entries in the matrix are Q-values that represent the expected future rewards of taking a specific action from a specific state.

In the learning process of action section, an agent must choose between exploring new actions or exploiting known rewarding actions. There are two to control by the epsilon-greedy strategy. An agent can explore the map or exploit the map. If the agent chooses exploration, it will randomly select an action to discover the maximum possibilities. Additionally, if the agent chooses exploitation, it will select the action with the highest Q-value to maximize the reward.

$$a = \begin{cases} randomaction & withprobability\epsilon \\ \mathrm{argmax}_a Q(s, a) & withprobability 1 - \epsilon \end{cases} \tag{8}$$

Third, the algorithm needs to design the rule of updating the Q-table. Q-learning updates the Q-values on the rewards received and the estimated future rewards. This is done using the Temporal Difference (TD) learning method. To use the TD learning method, the algorithm needs to define the TD target and TD error. TD target is the sum of the immediate reward and the discounted maximum future reward. TD error stands for the difference between the TD target and the current Q-value. Finally, the Q-value for state-action pair (s, a) is updated by the TD error. The update shifts the Q-value towards of TD target to TD error, effectively refining the agent's estimate of the long-term value of taking action a in state s.

$$\text{TDtarget} = r + \gamma Q\left(s', a'\right) \tag{9}$$

$$\text{TDerror} = \text{TDtarget} - Q\left(s, a\right) \tag{10}$$

$$Q(s, a) \leftarrow Q(s, a) + \alpha \times \text{TDerror} \tag{11}$$

The formula can be reduced into

$$Q(s, a) \leftarrow Q(s, a) + \alpha[r + \gamma \max_{a'} Q(s', a') - Q(s, a)] \tag{12}$$

In the training loop, first, the algorithm needs to setup initialization. For environment and hyperparameters, First, the algorithm needs to load the map. Second, the algorithm initializes with the loaded points and a randomly generated list of wall points. This sets up the environment in which the Q-learning agent will operate. Third, the algorithm needs to set the hyperparameters. ϵ is the exploration rate, which determines how often the agent will choose a random action instead of the best-known action. α is the learning rate, which controls how much new information overrides old information in the Q-table. γ is discount factor, which represents the importance of future rewards.

For Q-table initialization, the algorithm defines the number of states and actions in the environment obtained from the environment's observation and action spaces. The Q-table stores the Q-values for each state-action pair and is initialized to zeros.

For the metrics initialization, the algorithm saves eight attributes. Along with each case, in each episode, *rewards* to track the total reward obtained, *numSteps* to track the number of steps taken, *guardPositions* record the guard positions at the end.

Moreover, *explorationCount* and *exploitationCount* count how often the agent explores or exploits, *actionDistribution* count how often each action was taken, *stateVisits* count how often each state was visited, *avgRwards* and *movingAvg Rewards* track the average and moving average of reward, *avgActionValues*, *policyEntropy* and *actionValueVariation* for additional analysis of the policy and Q-value.

After the environment initialization, the algorithm starts the training loop. In the process, the episode loop will bag the loop of the steps. Episodes are the shell of the egg, and the loop of the steps is the yolk of the egg. The episode loop runs for a specified number of episodes; for each episode, the algorithm resets the environment and resets initializes episode variables. The environment reset stands for returning the initial state.

The steps loop is running for the agent to interact with the environment; each episode consists of up to 30 steps. In this loop, the algorithm will repeatedly work five actions.

First, the algorithm uses the action selection function; an action is chosen on epsilon-greedy strategy, is using the action selection function, which definite before.

Second, the algorithm uses a step function to run the task of action handling, field of view calculation, and reward calculation. The algorithm stores the value into three variables, *nextState*, *reward*, and *done*.

Third, the Q-table is updated using the Q-learning formula. The best next action is determined from the Q-table. The TD target and TD error are calculated. The Q-value for the current state-action pair is updated based on the TD error and the learning rate α.

Fourth, the log metric needs to be updated. The state is updated to *nextState*. *totalReward* is incremented by the received reward. *steps* is incremented. The action distribution and state visit counts are updated. The Q-values for the actions taken are accumulated in *episodeActionValues*.

Finally, if the *done* flag is set to true, the steps loop breaks, ending the episode early.

After all, the algorithm processes the post-episode phase. After each episode, the total reward and number of steps are appended to their respective lists. The Q-table is copied and stored for analysis. The guard positions at the end of the episode are recorded. Average rewards and moving averages are calculated and stored. The average action values are updated. Policy entropy, which measures the randomness of the policy, is calculated and stored. The variation in action values is also calculated and stored.

A. PSEUDOCODE

See Algorithm 1.

IV. EXPERIMENTS

In this section, the algorithm wants to prove the performance of Reinforcement Learning in the grid world. Moreover, experiment 4 displays the application of real-world showcase using this algorithm.

A. DIFFERENT GIRD SIZES IN POLYGON

This experiment aims to scale affection. To achieve this, the algorithm used the same polygon as the art gallery. Moreover, the algorithm used his algorithm to get the grid point with different scales. In this experiment, the algorithm chose a scale from 0.1 to 0.5 to obtain a number of each half-space. The algorithm makes those gird points as the map of an art gallery. After defining the environment of an art gallery, the algorithm started to run the reinforcement learning model. In this paper, our team chose Q-learning as the main algorithm. In this experiment, the algorithm logged the total rewards and policy entropy over each episode. The algorithm visualizes each result from belong scale.

The result shows the explored properties of this algorithm. While the scale is smaller, the public entropy gets lower, and the total reward gets bigger. As a result, the algorithm can handle more calculated space while the scale gets smaller.

Algorithm 1 Field of View Calculation

```
Require: points ∈ $\mathbb{R}^2$, guardPos ∈ points,
Ensure: visiblePoints
1:  visiblePoints ← ∅
2:  guardPos ← $(x_g, y_g)$
3:  function findParallelIndices(points, guardPos)
4:      vertical ← $\{(x, y)|x = x_g\}$
5:      horizontal ← $\{(x, y)|y = y_g\}$
6:      return (vertical, horizontal)
7:  end function
8:  function findDiagonalIndices(points, guardPos)
9:      diagonal ← $\{(x, y)||x - x_g| = |y - y_g|\}$
10:     leftBottom ← $\{(x, y) \in$ diagonal$|x < x_g \wedge y < y_g\}$
11:     leftTop ← $\{(x, y) \in$ diagonal$|x < x_g \wedge y > y_g\}$
12:     rightBottom ← $\{(x, y) \in$ diagonal$|x > x_g \wedge y < y_g\}$
13:     rightTop ← $\{(x, y) \in$ diagonal$|x > x_g \wedge y > y_g\}$
14:     return (leftBottom, leftTop, rightBottom, rightTop)
15: end function
16: function calculateFieldOfView(points, guardPos, walls)
17:     (leftBottom, leftTop, rightBottom, rightTop) ← findDiagonalIndices(points, guardPos)
18:     alignmentDict ← {vertical, horizontal, leftBottom, leftTop, rightBottom, rightTop}
19:     for dir ∈ alignmentDict do
20:         wallsInDir ← walls ∩ dir
21:         wallsInDirSorted ← sort(wallsInDir)
22:         closestWalls ← determineClosestWalls(wallsInDirSorted, guardPos)
23:         filteredPointsDir ← filterPointsBasedOnWalls(dir, closestWalls, guardPos)
24:         visiblePoints ← visiblePoints ∪ filteredPointsDir
25:     end for
26:     return visiblePoints
27: end function
28: function function determineClosestWalls(wallsSorted, guardPos)
29:     before ← $\{w|w <$ guardPos$\}$
30:     after ← $\{w|w >$ guardPos$\}$
31:     closestBefore ← max(before)
32:     closestAfter ← min(after)
33:     return (closestBefore, closestAfter)
34: end function
35:
36: function filterPointsBasedOnWalls(dir, closestWalls, guardPos)
37:     (closestBefore, closestAfter) ← closestWalls
38:     if closestBefore = ∅ then
39:         return $\{p|p \leq$ closestAfter$\}$
40:     else if closestAfter = ∅ then
41:         return $\{p|p \geq$ closestBefore$\}$
42:     else
43:         return $\{p|p \leq$ closestBefore $\vee\ p \geq$ closestAfter$\}$
44:     end if
45: end function
46: return calculateFieldOfView(points, guardPos, walls)
```

B. THE PARAMETER COMBINATE EXPERIMENT

This experiment showed the important layer between three parameters of q learning and complex parameters of maps. To challenge the ability of guard view and arrange, the algorithm added the wall in the polygon map. In this paper, the view arranged of the guard is defined as 8 sides. Obtain the diagonal. While the wall is stocked with the view line, the view line over the wall will be canceled. In this experiment, the algorithm combines the multiple value to find the most valuable to handle result quality. Figure 7 and Figure 8 show the different combinations in the same gird map. The value is followed by the mean of the same parameter. In total, each alpha, epsilon, and gamma have three different values. Hence, there are three bars for each value.

This is adding a wall range to diffuse the guard view. In this experiment, the algorithm takes the average value as the final score. The algorithm visualizes the result from Figures 9 to 10.

According to Figure 7 and Figure 8, while the map without walls, the alpha displays the key parameter, and the alpha can adjust the result most obviously. Similarly, according to figures 9 and 10, while adding the wall into maps, the

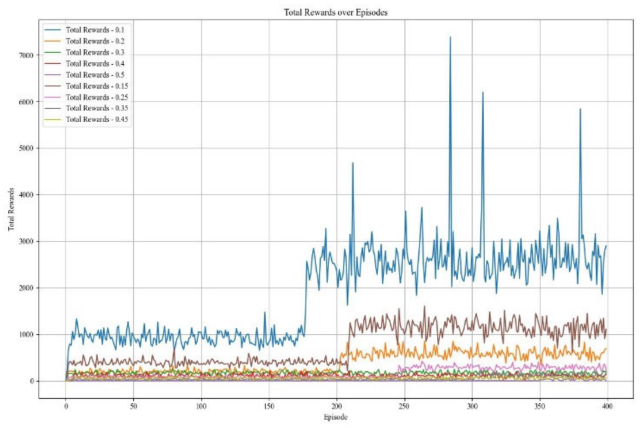

FIGURE 5. Total reward into each epoch. Each chose scale threads have different color. The blue line, 0.1 scale, have the highest reward valued. Meaning the 0.1 scale is the best.

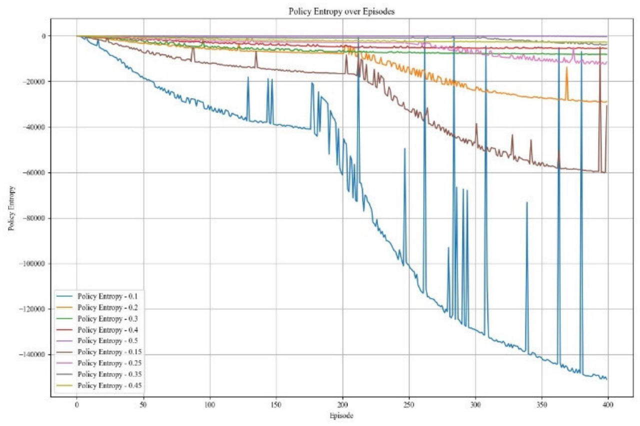

FIGURE 6. The public entropy of each epoch. Each chose scale threads have different color. The blue line, 0.1 scale, have the lowest entropy valued. Meaning the 0.1 scale is the best.

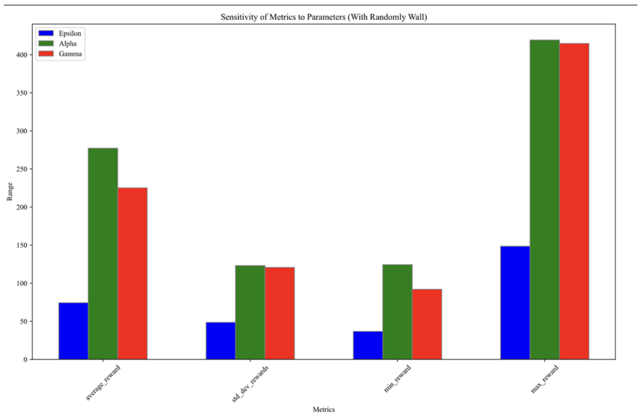

FIGURE 7. Four impact value upon different parameter value. Four pointers are average reward, standard dev reward, minimum reward and maximum rewards.

most key parameter turns to epsilon. This experiment can help people choose the adjust parameter in different situation. In summary, if the map is without a wall, with low complexity,

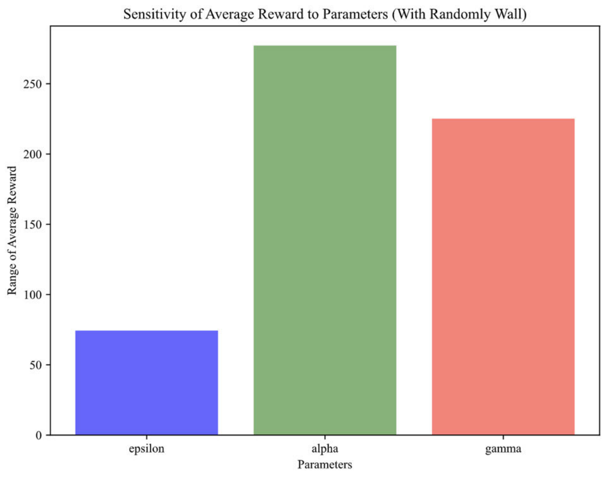

FIGURE 8. The mean average word into different parameter values. The purple bar is epsilon, the green bar is alpha, the pink bar is gamma. The alpha get the highest average reward.

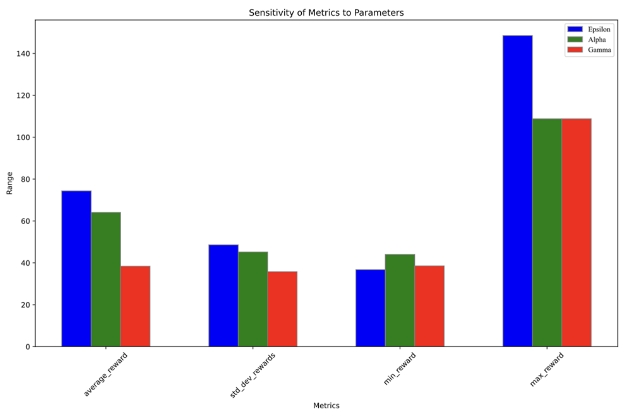

FIGURE 9. Four impact value upon different parameter value, adding chaos wall. Four pointers are average reward, standard dev reward, minimum reward and maximum rewards.

the best adjusted parameter is alpha. Differently, if the map has a wall with high complexity, the best adjust factor is epsilon.

C. THE PERFORMANCE OF DIFFERENT SCALES

This experiment stands for testing the generalized ability of the Q-learning algorithm. The algorithm chose four scales: 10, 100, 152, and 180. Moreover, the algorithm extracts the attribute from each polygon and aims to compare the complex scale of each environment.

Table 1 presents the attributes of various polygons. It can be classified into two trends: the complexity of polygons and the size of polygons. The complexity of a polygon obtains the number of vertices, compactness, and edge length standard deviation. The size of polygons is related to perimeter, area, circumscribed, and inscribed radius. In the complexity of polygon analysis, while the polygon is more complex, the attributes of the number of vertices, compactness, and

TABLE 1. Attribute of each polygon.

Scale	Perimeter	Area	Number of Vertices	Compactness	Edge Length Std Dev	Circumscribed Radius	Inscribed Radius
10	577.88	590.0	121	45.04	7.23	27.11	3.50
110	135,347.88	119,358.5	25,635	12,213.49	27.92	388.14	112.41
152	262,970.83	230,907.0	49,631	23,832.42	33.09	545.03	157.57
180	371,636.75	325,252.5	70,053	33,791.45	36.14	649.53	187.64

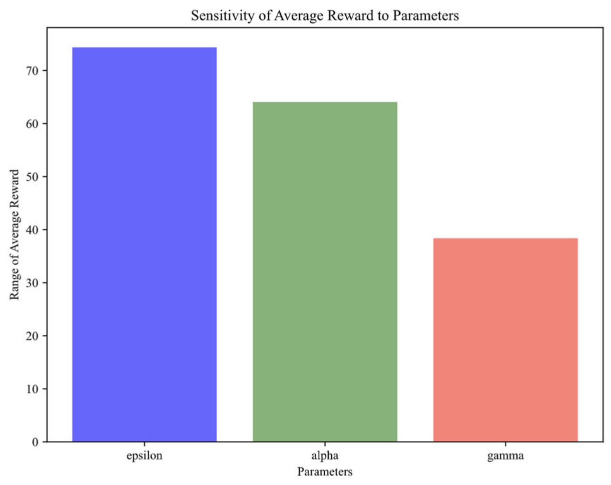

FIGURE 10. The mean value of average reward between different parameters, including wall account. The purple bar is epsilon, the green bar is alpha, the pink bar is gamma. The epsilon get the highest average reward.

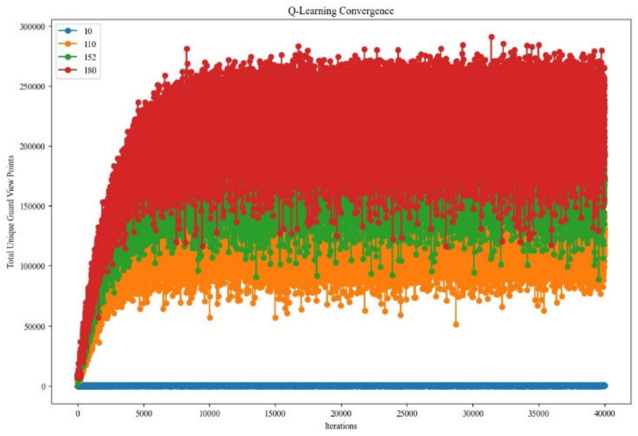

FIGURE 11. The Q-Learning value into different iterative. Obviously, while the scale of the map was bigger, the max q-value got bigger, too. The different color area means different scale. The red area, 180 scales, has the highest total guard viewpoints.

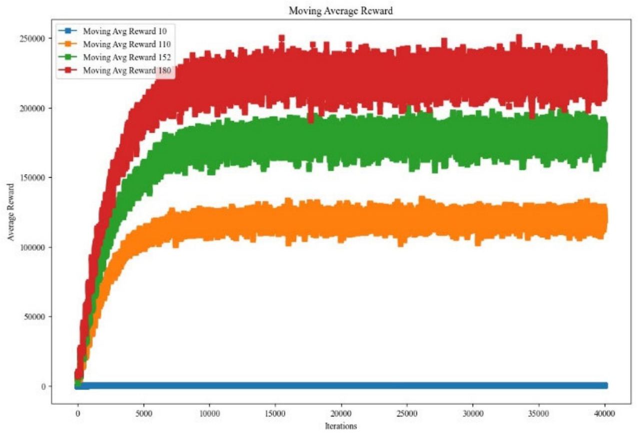

FIGURE 12. The mean value of the average moving reward between different scale maps includes a wall account. The different color area means different scale. The red area, 180 scales, has the highest mean value of the average moving reward.

edge length standard deviation are larger in more complex polygons. Also, in the size of polygon analysis, while the size of the polygon gets bigger, the attribute of perimeter, area, circumscribed radius, and inscribed radius will be larger.

In Table 1, the algorithm clearly finds the trend; while the scale gets bigger, the attribute's value gets larger. No matter which attribute, each one is following this trend. Consequently, for those polygons, while the scale is bigger, the complexity and size of the polygon are larger. For this reason, using these polygons to test the generalized ability of Q-learning is appropriate.

In this experiment, the algorithm is tested on maps of varying scales. Notably, as the map scale increases, the policy entropy decreases while the moving average reward increases. Similar to Experiment 1, these results demonstrate the algorithm's capability to adapt to different map scales. Furthermore, Figure 12 illustrates the mean average reward for each map over steps 5000 to 10000, indicating that the algorithm maintains its convergence properties even when processing maps of different scales.

D. THE SHOWCASE INTO THE REAL WORLD

The environment of this project is folium and osmnx, the toolkit to support catching real map data. In the first step, the algorithm randomly generates the points in a special area by the osmnx toolkit. The amount of points is limited by input; in this experiment, the algorithm took the number limit to 200. The second step is to select the node index. While the algorithm selected the special node index, the algorithm catches the node surface distance using geographic distance. The algorithm only saved the node within the designated distance range. In this experiment, our special node ID is 38651080, and the radius distance is 250 meters.

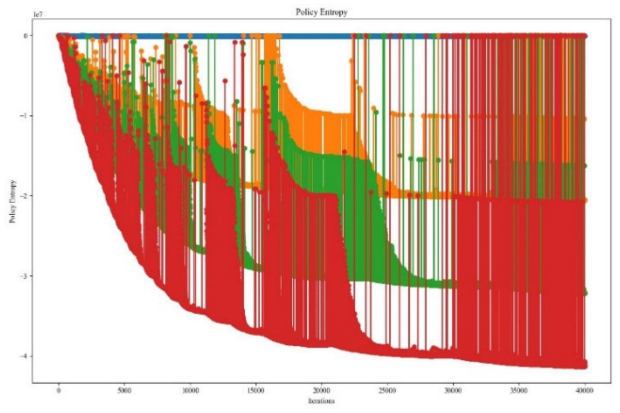

FIGURE 13. The policy entropy while each iterative. The red one is the biggest scale map. The blue one is the smallest scale map.

Step third was setting up nodes with random quantities. Similarly, for each edge in the graph, the function retrieves its geometry. And keep on the extract endpoint distance of the edge from the designated point to the edge end. After that, the algorithm uses the filtering logic to clip out. If the edge is within the range, or at least one of its endpoints is within the defined distance from the special node, the algorithm picks it up and saves it. While confirming the range, the algorithm visualizes all the edges and nodes saved into the HTML map.

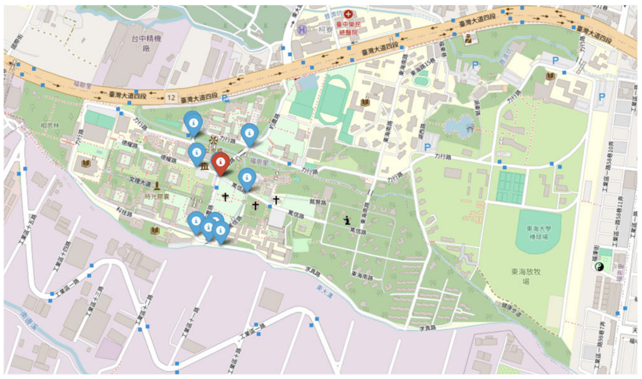

FIGURE 14. The map of Tunghai University. In this experiment, using this map as the based background.

The fourth step involves acquiring the node shape and loading the nodes within range. The x and y-axis values are then used to create the polygon. The algorithm employs the convex hull method to identify the outermost boundary of the polygon. Once the locally optimized polygon is obtained, the algorithm applies the method from [4] to transform the polygon into grid points. This polygon represents the extracted hole geometry points within the map. Finally, the polygon data is saved into a temporary file.

The fifth step is to clear duplicate edges. In step three, the algorithm saved the edge's point index and geometry coordinate. The point is iterative. Each edge, by the ratio, randomly generates the points. The points coordinate is calculated proportionally between the starting and ending points

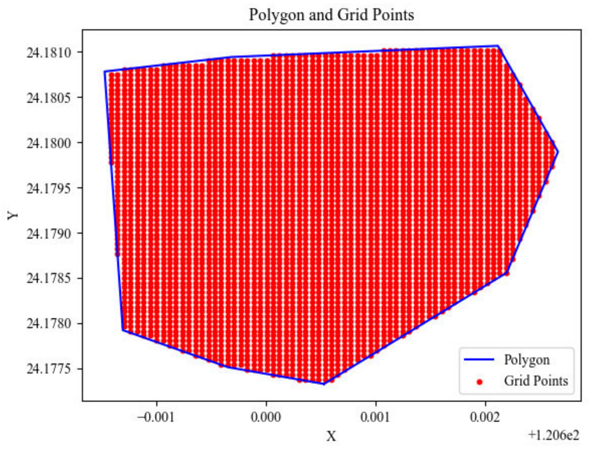

FIGURE 15. The convex hull surrounds the map and includes a circle. The red points be denoted as the pointed into the polygon, could be used for art gallery problem.

of the edge. The core issue to clear duplicate edges, is the decimal control. In the geometry coordinate, two points will not duplicate. If the algorithm needs to extract the points, it has to round off this coordinate by discarding some decimal places.

The digital process is rounding each coordinate point to the specified number of decimal places, in this experiment, setting the specified number as 4. After rounding, iterative each rounding point; if the points are in the polygons data, save this point to road_maps.csv; if not in polygons data, save this point to building_places.csv. While the iterative is done, the algorithm gets the initial data for the road and building. To train reinforcement learning, the algorithm adding the process is the standardization of both data. After standardization, the algorithm can visualize the data, road, and building. Also, additionally, the training dataset is already prepared.

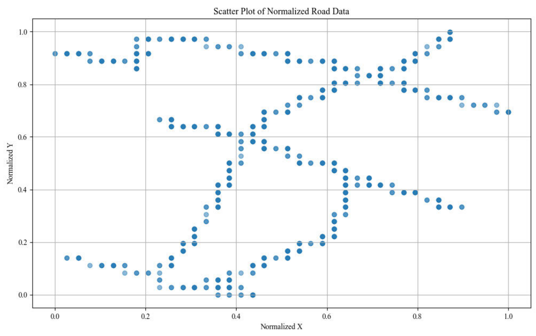

FIGURE 16. The road maps grid points extracted by experiment process.

The sixth step is putting the training dataset into reinforcement learning. Before loading the dataset to the RL algorithm, needs to decide the parameter values. The alpha and gamma parameters are set as 0.9. The epsilon parameter is set as 0.1. The Max iterative is set as 1000.

The results show the placement of each guard. Both maps, roads, and buildings have high performance. The q-value

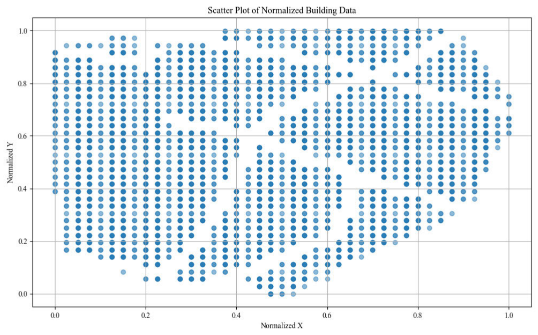

FIGURE 17. The building places grid points extracted by experiment process.

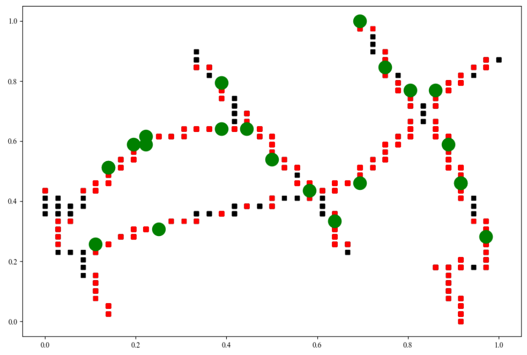

FIGURE 18. The map will optimize the position of sensor placement and road situation. The green points is the placement of guard, the red points is the view of guard. In this experiment, the red point almost covered the full map.

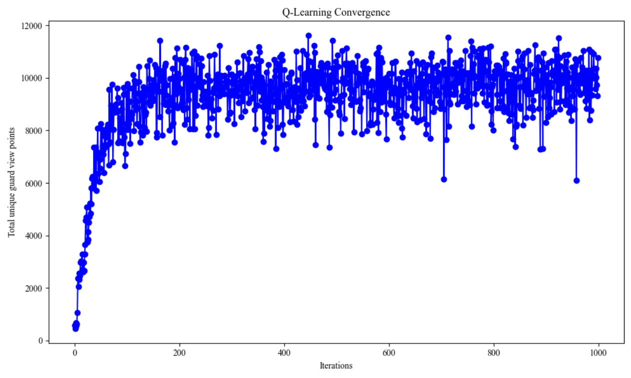

FIGURE 19. The q-value converges of road maps into 1000 runs. As the iterative getting more and more, the tread be stably release.

results indicate the stability and convergence of the algorithm in finding solutions. In summary, this algorithm has high performance in real-world showcases. It has high confidence to use it in different real-world maps.

E. COMPARE THE ALGORITHM PERFORMANCE

This experiment aimed to compare the problem-solved ability between Q-Learning (QL), BGreedy Heuristic (GH) [4], and exact Integral Linear Programming (ILP) [4]. For the best

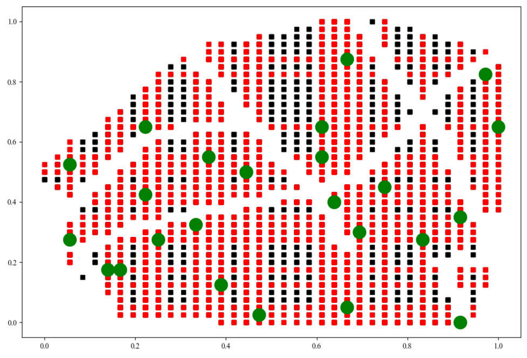

FIGURE 20. The map with optimize position of sensor placement. The green points is the placement of guard, the red points is the view of guard. In this experiment, the red point almost covered the full map.

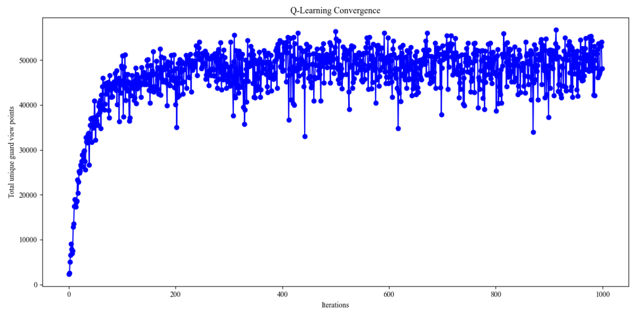

FIGURE 21. The q-value converges of building sensor placement optimized into 1000 runs. As the iterative getting more and more, the tread be stably release.

strategy, meaning that the algorithm is the best solution to solve the art gallery problem.

The benchmark of this experiment is from [30]; the FAT Instances benchmark includes polygons of large areas and wide interiors, which could test the performance of an art gallery problem-based algorithm.

The number of guards selected, Bthe total view of the guards in the final solution, and the running times are the main pointers to estimate algorithm performance. The objective function of the art gallery problem is the maximum total view of the guards, minimum number of guards, and running times of the experiment.

In Table 2 and Table 4, Size is the instance of the benchmark. Algo is the name of the algorithm. Num is the point size of this map. QRT is Q-learning iterative times. Sol is the final selection of guards. Opt is the total views of the final guards. Times is the running time second of the algorithm worked.

From Table 2, except for the map of 10 sizes, QL gets the minimum number of guards. The ILP gets the maximum total view of the guard. However, the mean absolute deviation of the total view between QL and ILP is 2; this is an acceptable range. The minimum running times of GH but QL stably consist of the same times in each map. ILP would take a very long time to calculate the map; in a map with 16 sizes, ILP needs 1813.86603 seconds to calculate the map. This

TABLE 2. The performance between Q-Learning (QL), BGreedy Heuristic (GH) [4] and exact Integral Linear Programming (ILP) [4] in a small map.

Size	Algo	QRT	Num	Sol	Opt	Times
10	QL	1000	121	12	120	2.18688
	GH			10	121	0.38489
	ILP			9	121	0.29424
12	QL	1000	198	10	196	2.15848
	GH			12	198	1.11087
	ILP			11	198	1.45655
14	QL	1000	284	12	283	2.93884
	GH			16	284	2.57357
	ILP			14	284	100.74570
16	QL	1000	394	14	390	2.41421
	GH			20	394	4.94099
	ILP			16	394	1813.86603

is too heavy and too big in calculate times. Therefore, this experiment added a larger map to test the stability of QL and GH.

TABLE 3. The mean value from Table 2 between three algorithms, Q learning, greedy heuristic, and integral linear programming.

Algo	Sol	Opt	Times
QL	12.0	247.25	2.424602
GH	14.5	249.25	2.252580
ILP	12.5	249.25	479.090630

From Table 3, QL gets the minimum number of selected guards. GH gets the minimum running times. But the time between GH and QL is 0.172022; it needs to be experimented on a larger map. The ILP and GH get the maximum total view of the guard, but the ILP mean running times are too big to calculate. Meaning that ILP is too heavy to experiment on a larger map. Therefore, the experiment tested the QL and GH on a larger map; the result is logged in Table 4.

TABLE 4. The performance between Q-Learning (QL), BGreedy Heuristic (GH) [4] and exact Integral Linear Programming (ILP) [4] in a large map.

Size	Algo	QRT	Num	Sol	Opt	Times
30	QL	10000	1647	12	1642	5.10448
	GH			41	1647	129.19031
50	QL	10000	4953	14	4951	3.99045
	GH			74	4953	1929.76290
70	QL	10000	10054	16	10036	19.44581
	GH			109	10054	13009.42130
90	QL	10000	16945	14	16933	28.23813
	GH			144	16945	50866.77791
110	QL	10000	25635	16	25549	57.28683
	GH			175	25635	149241.7619

In Table 4, while the map is bigger, the QL gets the minimum number of guards and minimum running time. In total view of the guard, GH gated the maximum value in each map. But the total view of the guard between QL and GH is the close number. The max gap of total views of final guards is a map with size 110. The gap is 82, and it is only 0.031% in the 25635 sizes. So, the gap is an acceptable range.

TABLE 5. The mean value from Table 4 between two algorithms, Q learning and greedy heuristic.

Algo	Sol	Opt	Times
QL	14.4	11822.2	21.777827
GH	108.6	11846.2	62795.894405

From table 5, Q-learning (QL) is a better choice for the art gallery problem compared to the Greedy Heuristic (GH) for several reasons. First, QL achieves a much lower mean solution value (14.4) compared to GH (108.6), indicating it finds more optimal solutions. Second, QL requires significantly less execution time (21.77) than GH (62795.89), highlighting its efficiency, especially for larger problems. Although QL's mean total view of guard (Opt) is lower than GH, the high performance of Sol and running times is the strong factor in choosing QL as a better algorithm. Overall, QL performs faster and provides better solutions, making it a superior choice for this type of problem.

In conclusion, Q-Learning (QL) is the best algorithm for solving the art gallery problem compared to Greedy Heuristic (GH) and Integral Linear Programming (ILP). QL consistently selects fewer guards while achieving a total view close to ILP's optimal solution. For example, for a map size of 12, QL selects 10 guards, while GH selects 12 and ILP 11. QL's total view of 196 is only 2 less than ILP's 198.

QL also outperforms GH and ILP in running time. For map size 50, QL runs in 3.99 seconds, while GH takes 1929.76 seconds and ILP is slower for smaller maps like 16 (1813.87 seconds). Even for large maps like size 110, QL achieves a total view of 25,549, just 82 less than GH's 25,635, a negligible difference (0.031%).

Therefore, QL is the most efficient and reliable solution, especially for larger problem instances, offering a good balance between guard count, total view, and running time.

V. DISCUSSION

From experiment 1, the result shows the different total reword and entropy while using different gird coarsening sizes. Basically, the blue line has a huge leading score in Figures 5 and 7. The blue line is the 0.1 scale of grid worlds. This result stands for the smallest scale between 0.1 to 0.5, which can achieve the most development space while the q-learning algorithm is working. Moreover, the line value in Figure 5 and Figure 6, is decreases as the scale decreases. As a consequence, the achieve development space is growth as the scale decreases.

Experiment 2 tests the most valuable factor to impact the adventure process. Figures 7 and 8 show the fixed value of the wall account. Meanwhile, Figures 9 and 10 are more complex structures. Figures 9 and 10 are based on the experiment for testing parameters of compose alpha, epsilons, and gamma. At the same time, this part, adding three wall account values, aims to test generalizability. According to Figure 7 and Figure 8, the most important factor is alpha. In the case of Figures 9 and 10, the most important factor changes to epsilons. By all means, while running the same map, changing the alpha value can get the clearest move of the result. Whatever, to experiment with different maps, changing epsilons is the most significant to the development of different results.

Experiment 3 shows the general ability to q-learning into different scale maps. The difference between experiments 2 and 3 is that experiment 2 adds multiple accounts of the wall. The experiment has put the guard into multiple scales of the map, which fixed the account of the wall. Figures 11 to 13 show the q-value, moving average award value, and public entropy of each iterative. Significantly, the Q-learning algorithm can release the value to fixed at 10000 iterative times. After 10000 iterative times, the Q-learning algorithm is focused on releasing public entropy. Moreover, the performance of growth belongs to scale value. In summary, Q-learning shows powerful release astringency into different scales map. Furthermore, the map with a bigger scale can get more adventure space for an optimized guard view.

Experiment 4 fully displays the performance of the algorithm working in a real-world showcase. The results show the algorithm can handle the map data in calculation and also indicate the stability and convergence of the algorithm. As a result, the method of grid system reinforcement learning for art gallery problems demonstrated strong adaptability across maps of different scales. At the same time, the method indicates reliable results in real-world showcases.

Experiment 5 compares the Q-Learning (QL), BGreedy Heuristic (GH) [4], and exact Integral Linear Programming (ILP) [4]. The art gallery problem aimed to maximize the total view of guards and minimize the number of guards and the running times. From the small map experiment, in table 2 and 3, Q-learning (QL) is the best choice because it achieves the lowest number of selected guards (12.0) with a reasonable running time of 2.42 seconds, compared to ILP, which also selects 12.5 guards but takes significantly longer (479.09 seconds), and Greedy Heuristic (GH), which selects more guards (14.5) but only reduces running time slightly (2.25 seconds).

Moreover, from the big map experiment, in table 4 and 5, Q-Learning (QL) is the best choice for large map problems, as it outperforms Greedy Heuristic (GH) in solution quality and computation time, with QL achieving an average of 14.4 solutions and 21.77 seconds per map compared to GH's 108.6 solutions and 62795.89 seconds, as shown in Table 5. Therefore, experiment 5 proved while choosing the algorithm between Q Learning (QL), Greedy Heuristic (GH) [4], and Integral Linear Programming (ILP) [4], Q Learning is the best algorithm to solve the NP problem.

VI. CONCLUSION

In this study, we proposed a grid-based reinforcement learning approach to address the Art Gallery Problem (AGP), demonstrating its effectiveness and efficiency through extensive experiments. The results show that our method achieves significant improvements over traditional techniques such as Greedy Heuristic (GH) and Integral Linear Programming (ILP) in terms of guard selection, computation time, and solution quality.

Through a series of experiments, we evaluated the proposed method across different grid scales, parameter settings, and map complexities. Key findings include:

1. Smaller grid scales (e.g., 0.1) provide enhanced development space for Q-learning, as shown in Experiments 1 and 3.
2. Parameters like alpha and epsilon significantly impact the learning process, with their influence varying based on map structure and scale, as detailed in Experiment 2.
3. The algorithm exhibits strong generalization capabilities across maps of varying sizes and complexities, achieving rapid convergence and stable performance (Experiment 3).
4. When applied to real-world scenarios, the proposed method maintains adaptability and robustness, handling diverse map data effectively (Experiment 4).
5. In direct comparison with GH and ILP, Q-learning consistently produces better solutions with fewer guards and faster computation times, particularly for large-scale maps, as evidenced by Experiment 5.

In conclusion, the grid-based reinforcement learning approach not only outperforms traditional methods in solving the AGP but also demonstrates scalability and adaptability for real-world applications. These results suggest that reinforcement learning is a promising direction for addressing NP-hard problems like the AGP, combining computational efficiency with high solution quality. Future work could explore further optimizations and adaptations of this method for other complex spatial optimization problems.

REFERENCES

[1] D. Lee and A. Lin, "Computational complexity of art gallery problems," *IEEE Trans. Inf. Theory*, vol. IT-32, no. 2, pp. 276–282, Mar. 1986, doi: 10.1109/TIT.1986.1057165.

[2] T. C. Shermer, "Recent results in art galleries (geometry)," *Proc. IEEE*, vol. 80, no. 9, pp. 1384–1399, Sep. 1992, doi: 10.1109/5.163407.

[3] M. Abrahamsen, A. Adamaszek, and T. Miltzow, "The art gallery problem is ∃ℝ-complete," *J. ACM*, vol. 69, no. 1, pp. 1–70, Feb. 2022, doi: 10.1145/3486220.

[4] M. Predojević, M. Đukanović, M. Grbić, and D. Matić, "Can greedy-like heuristics be useful for solving the weighted orthogonal art gallery problem under regular grid discretization?" *Int. J. Electr. Eng. Comput.*, vol. 5, no. 2, Jan. 2022, doi: 10.7251/ijeec2102077p.

[5] U. Stańczyk and B. Zielosko, "Heuristic-based feature selection for rough set approach," *Int. J. Approx. Reasoning*, vol. 125, pp. 187–202, Oct. 2020, doi: 10.1016/j.ijar.2020.07.005.

[6] J. Li, Z. A. Talebi, R. Chirkova, and Y. Fathi, "A formal model for the problem of view selection for aggregate queries," in *Advances in Databases and Information Systems*, 2005, pp. 125–138, doi: 10.1007/11547686_10.

[7] S. Chaudhuri, R. Krishnamurthy, S. Potamianos, and K. Shim, "Optimizing queries with materialized views," in *Proc. 11th Int. Conf. Data Eng.*, 1995, pp. 190–200, doi: 10.1109/icde.1995.380392.

[8] L. P. Kaelbling, M. L. Littman, and A. W. Moore, "Reinforcement learning: A survey," *J. Artif. Intell. Res.*, vol. 4, pp. 237–285, May 1996, doi: 10.1613/jair.301.

[9] R. H. Crites and A. G. Barto, "Elevator group control using multiple reinforcement learning agents," *Mach. Learn.*, vol. 33, no. 2/3, pp. 235–262, 1998, doi: 10.1023/a:1007518724497.

[10] P. Osborne, H. Nomm, and A. Freitas, "A survey of text games for reinforcement learning informed by natural language," *Trans. Assoc. Comput. Linguistics*, vol. 10, pp. 873–887, Aug. 2022, doi: 10.1162/tacl_a_00495.

[11] B. Porr and F. Wörgötter, "Learning with 'relevance': Using a third factor to stabilize Hebbian learning," *Neural Comput.*, vol. 19, no. 10, pp. 2694–2719, Oct. 2007, doi: 10.1162/neco.2007.19.10.2694.

[12] R. Chirkova, A. Y. Halevy, and D. Suciu, "A formal perspective on the view selection problem," *VLDB J. Int. J. Very Large Data Bases*, vol. 11, no. 3, pp. 216–237, Nov. 2002, doi: 10.1007/s00778-002-0070-0.

[13] S. P. Fekete, S. Friedrichs, A. Kröller, and C. Schmidt, "Facets for art gallery problems," *Algorithmica*, vol. 73, no. 2, pp. 411–440, Oct. 2015, doi: 10.1007/s00453-014-9961-x.

[14] F. Hoffmann, M. Kaufmann, and K. Kriegel, "The art gallery theorem for polygons with holes," in *Proc. 32nd Annu. Symp. Found. Comput. Sci.*, 1991, pp. 39–48, doi: 10.1109/sfcs.1991.185346.

[15] B. Alidaee, G. A. Kochenberger, and M. M. Amini, "Greedy solutions of selection and ordering problems," *Eur. J. Oper. Res.*, vol. 134, no. 1, pp. 203–215, Oct. 2001, doi: 10.1016/s0377-2217(00)00252-6.

[16] Y. Akçay, H. Li, and S. H. Xu, "Greedy algorithm for the general multidimensional knapsack problem," *Ann. Oper. Res.*, vol. 150, no. 1, pp. 17–29, Feb. 2007, doi: 10.1007/s10479-006-0150-4.

[17] B. Jang, M. Kim, G. Harerimana, and J. W. Kim, "Q-learning algorithms: A comprehensive classification and applications," *IEEE Access*, vol. 7, pp. 133653–133667, 2019, doi: 10.1109/ACCESS.2019.2941229.

[18] H. Murao and S. Kitamura, "Q-learning with adaptive state segmentation (QLASS)," in *Proc. IEEE Int. Symp. Comput. Intell. Robot. Autom., Towards New Comput. Princ. Robot. Autom.*, Jul. 1997, pp. 179–184, doi: 10.1109/CIRA.1997.613856.

[19] S. Spanò, G. C. Cardarilli, L. Di Nunzio, R. Fazzolari, D. Giardino, M. Matta, A. Nannarelli, and M. Re, "An efficient hardware implementation of reinforcement learning: The Q-learning algorithm," *IEEE Access*, vol. 7, pp. 186340–186351, 2019, doi: 10.1109/ACCESS.2019.2961174.

[20] C. Bingane, "Tight bounds on the maximal perimeter and the maximal width of convex small polygons," *J. Global Optim.*, vol. 84, no. 4, pp. 1033–1051, Dec. 2022, doi: 10.1007/s10898-022-01181-9.

[21] L. Barba, L. E. Caraballo, J. M. Díaz-Báñez, R. Fabila-Monroy, and E. Pérez-Castillo, "Asymmetric polygons with maximum area," 2015, *arXiv:1501.07721*.

[22] Y. Chen, "Derivation of the functional relations between fractal dimension of and shape indices of urban form," *Comput., Environ. Urban Syst.*, vol. 35, no. 6, pp. 442–451, Nov. 2011, doi: 10.1016/j.compenvurbsys.2011.05.008.

[23] Y. Chen, A. Janowczyk, and A. Madabhushi, "Quantitative assessment of the effects of compression on deep learning in digital pathology image analysis," *JCO Clin. Cancer Informat.*, vol. 2, no. 4, pp. 221–233, Nov. 2020, doi: 10.1200/cci.19.00068.

[24] W.-D. Richter and K. Schicker, "Circle numbers of regular convex polygons," *Results Math.*, vol. 69, nos. 3–4, pp. 521–538, Jun. 2016, doi: 10.1007/s00025-016-0534-y.

[25] L. F. Tóth, "Approximation by polygons and polyhedra," *Bull. Amer. Math. Soc.*, vol. 54, no. 4, pp. 431–438, Oct. 2017, doi: 10.1090/s0002-9904-1948-09022-x.

[26] P. Pech, "Computations of the area and radius of cyclic polygons given by the lengths of sides," in *Proc. Int. Workshop Automated Deduction Geometry*, Jan. 2004, pp. 44–58, doi: 10.1007/11615798_4.

[27] H. W. Chase, P. Kumar, S. B. Eickhoff, and A. Y. Dombrovski, "Reinforcement learning models and their neural correlates: An activation likelihood estimation meta-analysis," *Cognit., Affect., Behav. Neurosci.*, vol. 15, no. 2, pp. 435–459, Jun. 2015, doi: 10.3758/s13415-015-0338-7.

[28] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba, "OpenAI gym," 2016, *arXiv:1606.01540*.

[29] H. Xu, X. Miao, J. Li, M. Fan, L. Wang, H. Zheng, and C. Wen, "Mondrian virtual art gallery," in *Proc. Int. Conf. Virtual Reality Visualizat. (ICVRV)*, Oct. 2018, p. 136, doi: 10.1109/ICVRV.2018.00044.

[30] A. L. Bajuelos, A. P. Tomás, and F. Marques, "Partitioning orthogonal polygons by extension of all edges incident to reflex vertices: Lower and upper bounds on the number of pieces," in *Proc. ICCSA*. Berlin, Germany: Springer, Jan. 2004, pp. 127–136.

YUAN-HSUN LIAO received the M.Eng. degree from Tunghai University, Taichung, Taiwan, in 2006, and the Ph.D. degree in computer science from National Chung Cheng University, Chia-Yi, Taiwan, in 2013. Since 2019, he has been an Assistant Professor with Tunghai University. His research interests include artificial intelligence, machine learning, image processing, virtual reality, and e-learning.

PO-CHUN CHANG was born in Taiwan. He is currently pursuing the B.S. degree in computer science with Tunghai University, Taichung, Taiwan. His research interests include graph theory, topology, computation geometry, deep learning, big-scale quantum search, multimodal learning, large language model, optimize algorithm, and accelerating expansion of the universe.

HSIAO-HUI LI received the Ph.D. degree in information management from National Chung Cheng University, Chia-Yi, Taiwan, in 2013. Since 2022, she has been an Assistant Professor with the National Kaohsiung University of Science and Technology. Her research interests include technology-enhanced learning, education application, health care information systems, and machine learning.