修　士　学　位　論　文

Sub-optimally Solving the Multiple Watchman Route Problem

(複数の警備員によるグリッド警備問題に対するアルゴリズム)

令和4年度

東京大学大学院　総合文化研究科
広域科学専攻　広域システム科学系
学生証番号　31-216835

劉　冠廷

指導教員　福永アレックス

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Imagine you are a guard of a museum and you need to patrol the interior of the museum to ensure the safety of all the exhibits on the floor. To do so, you want to minimize your patrol path to increase the patrol frequency and you should be able to see all items in rooms on each path. This problem is called the Watchman Route Problem (WRP), where the task is to find a route that 'sees' every point in the environment. WRP was proven to be NP-hard for polygons [2] and has been addressed with approximate and optimal algorithms [29] [17] [25]. In many cases, due to the large area, it is unrealistic to patrol the entire museum with only one guard. Naturally, we would like to assign multiple guards to complete the patrol through division of work and cooperation. The task requires finding multiple routes so that every point inside the polygon is visible from at least one of the routes. This multi-person version of WRP is called the Multiple Watchman Route Problem (MWRP). It is motivated by many applications with elements of security and surveillance (e.g., guarding, exploring and analyzing buildings and areas), efficient usage of energy and time (e.g., it takes fewer frames to photograph on area), and more. In this problem, as the number of guards is usually predetermined, the measure of the result is often the route lengths. Extending the problem to support holes inside the polygon (i.e. obstacles inside museum) makes the problem more challenging.

In this thesis, we consider the problem of finding *k-watchmen routes*, where $k \geq 1$, in a polygon with holes such that multiple watchmen can transmit messages inside. There are two objectives which has been considered by previous work on the WRP: minimizing the maximum length of any route (MinMax) and minimizing the sum of the route length (MinSum). Both of these variants are known to be NP-hard [20]. The focus on this thesis is on the MinMax criterion, which aims to balance the watchmen's route lengths and minimizes the makespan of the routes. Packer et al. [21] proposed a heuristic method for MWRP in the polygonal domain possibly with holes. The algorithm is based on finding the minimum spanning tree of the pairwise shortest paths between guards, and then modifying the tree by substituting vertices and

removing redundant ones to create the routes. However, this algorithm has been used in problems with only unrestricted visibility, which limits its practical applications. Faigl [6] proposed a decoupled method to solve MWRP by dividing it into two independent problems: the Art Gallery Problem (AGP) and the Multiple Travelling Salesman Problem (MTSP). And combine the solutions of these two problems to solve MWRP. Faigl also presented a Self-Organizing Map (SOM) method [5] for MWRP and the performance of the SOM method is compared with the decoupled method (AGP+MTSP). However, these methods still have disadvantages. More details and discussion about the SOM and AGP+MTSP methods can be found in Section 3.2.

Formulating MWRP as a search problem on a grid-map, we propose a new method for computing suboptimal k watchmen routes that cover a grid map and satisfy the MinMax criterion for MWRP. This method involves two stages. In the first stage, we use a multi-class clustering algorithm to partition all cells of the grid into different groups (clusters). In the second stage, we use a generate-and-repair search algorithm to solve MWRP. At the route generation level, a WRP solver is applied to each watchman to find a watchman path within each cluster. In the route repair level, we use an iterative repairing search algorithm to repeatedly repair the paths until no further improvements can be made, in order to maximize the solution quality.

Our originality and contribution: Our proposed method for MWRP differs from previous work in that unlike previous work which assumed specific visibility constraints, our approach is independent of the visibility constraint. In addition, while previous approaches to the MWRP works required the solution to be a cycle, our approach does not necessarily generate cyclic solutions, and the routes can stop at any cell that satisfies the termination condition. We conduct an extensive experimental analysis of our method's performance, using optimal solutions to the single-agent WRP as a baseline for comparison. Our method appears to perform well on many small to medium-sized maps when compared to this baseline.

Limitations of our method: However several limitations remain in our method. Firstly, the method can take too much time on some large maps. Secondly, The number of node generations can be excessive when the map owns a large open area. Finally, the approach is limited to the MinMax objective, and it is unclear whether it will perform as effectively for a MinSum objective as it does for a MinMax objective.

The rest of this paper is organized as follows: the next chapter provides background information and details about the WRP solver that we selected, including terminology, heuristics, and optimization techniques. In Chapter 3, we provide a detailed definition of MWRP and review related work on MWRP. In Chapter 4, we describe the implementation of our proposed method. In Chapter 5, we present an experimental evaluation of the proposed MWRP algorithms. In Chapter 6, we summarize the contributions of the thesis and discuss potential improvements.

Chapter 2

Background and Related Work

2.1 A* Search

A* [11] is a graph search algorithm which is widely used in many fields of computer science. Starting from a specific starting vertex of a graph, it aims to find a path to the given goal vertex with the smallest cost.

A* maintains two sets of nodes called *OPEN* and *CLOSED*. *OPEN* is a priority queue which manages the nodes to be expanded, and *CLOSED* (often implemented as a hash table) stores nodes which have already been expanded.

A* explores the search space by repeatedly selecting u, a highest priority node from *OPEN*, generating the successors (neighbors) of u, and inserting the successors in u according to the following evaluation function. Let $g(u)$ be the cost of the path from the start node to u, and let $h(n)$ be an estimate of the path cost from u to a goal node. The evaluation function $f(u) = g(u) + h(u)$ is used to prioritize nodes in *OPEN* if $h(u)$ is a lower bound on the cost of any path from u to a goal, then $h(u)$ is called an admissible heuristic function.

In each iteration of A*, the lead node u is extracted from *OPEN* to check if it satisfies the goal condition. If it is, the path (solution) $path(u)$ that from the start node to u is returned and the main loop ends. If not, node u is expanded and its children nodes $children(u)$ are generated, evaluated using the evaluation function $f(u)$, and inserted into *OPEN*. While inserting each node $c \in children(u)$, the existence of c in *CLOSED* is checked. If c is not in *CLOSED*, it will be added to *OPEN* directly. If c is in *CLOSED*, if $g(c)$ is less than the g-cost of the previous entry in *CLOSED*, then the c is reinserted into *OPEN*. Otherwise, c is discarded.

2.2 K-means Clustering Algorithm

K-means [19] is the most commonly used clustering algorithm, that aims to partition n data points into k clusters. With K-means, each data point is assigned to a cluster such that the partition achieves a locally minimal value for mean square error (MSE):

$$MSE = \sum_{j=1}^{k} \sum_{X_i \in C_j} \frac{||X_i - C_j||^2}{n} \tag{2.1}$$

where X_i denotes data point locations and C_j denotes the centroid locations. The standard k-means algorithm consists of two repeatedly executed steps:

Assignment Step: Assign the data points to clusters by the nearest centroid:

$$P_j^{(t)} = \{X_i : ||X_i - C_j^{(t)}||^2 \leq ||X_i - C_{j^*}^{(t)}||^2 \quad \forall j^*, 1 \leq j^* \leq k\} \tag{2.2}$$

Update Step: Calculate the mean of each cluster:

$$C_j^{(t+1)} = \frac{1}{|P_j^{(t)}|} \sum_{X_i \in P_j^{(t)}} X_i \tag{2.3}$$

Where $P_j^{(t)}$ denotes the j-th cluster in the t-th iteation. The steps are repeated until centroid locations do not change. The solution converges to a local optimum, without a guarantee of global optimality. Therefore more complex k-means variants like k-means++ [1] have been proposed to try to obtain better results.

2.3 Watchman Route Problem

In the Watchman Route Problem (WRP), the task is to find a path for a traveling agent (watchman) on a map such that any location on the map is seen from at least one location on the path. the sight of the agent is modeled using a Line-of-Sight (LOS) function.

In our thesis, MWRP will be formulated as a search problem. As the basis for MWRP, WRP is also formulated as a search problem and with a heuristic search. We use the representation from Skyler et al 2022 [26] but modified some specific terms to avoid confusion.

The input of WRP is the tuple $\langle M, start, LOS \rangle$, where $M(V, E)$ is a grid-map, $start \in V$ is the start location of the watchman, and LOS is the line-of-sight function ($LOS : V \rightarrow L \subset V$). The LOS function determines the set of cells visible from each cell. The common LOS functions on grid-map are the cardinal lines (4-way) and diagonal lines (8-way). Note that like 4-way LOS, it can observe all things in cardinal direction with no distance limit unless it is blocked by an obstacle. For grid-map, blocked cells (non-traversable) are denoted as *obstacles* and V is

composed of the empty cells (traversable). We say (x, y) is an edge$\in E$ iff there is a legal move from x to y (and visa versa). We assume that only the four cardinal moves are legal actions to move along the grid. However, generalizing this work to allow diagonal moves or other moves should be straightforward.

A path $\pi = \langle p_0 = start, ..., p_n \rangle$ is a sequence of neighboring cells starting from *start*. The task is to find a watchman path in the map. In a $watchman\ path\ \pi$, for every cell $c \in V$ there is line-of-sight from at least one cell $p_i \in \pi$. The objective of WRP is to find a watchman path with minimal cost and we assume that the watchman does not have to return to the start location. In other words, the objective function is $cost(path)$. The important terms of the problem are defined below:

- State: A state is a pair $\langle location, seen \rangle$, where *location* is a cell (current location of the watchman) and *seen* is a list of cells that have been seen so far by the watchman.
- Node: A node S in the search space contains a *state*, a parent pointer, and a list of pointers to its children.
- Root: *Root* is a node such that $Root.location = start$ and $Root.seen = LOS(start)$
- Goal: Any node S where $S.seen = V$. Thus $Goal.location$ may be any cell$\in V$ as long as all $S.seen = V$
- Path and Cost: Every node S in the search space is associated with a path $\pi = \langle p_0 = start, ..., p_n \rangle$ which is determined by the sequence of edges in the search space that leads to S. This enables the pruning of duplicate nodes that have the same state. The cost of node S is the sum of the edge costs of the path.
- Priority Evaluation Function: To sort the nodes in the open list *OPEN*, $f(S) = g(S) + h(S)$ is used to determine the expansion order during search process. $g(S)$ is the cost of node S, and $h(S)$ is the heuristic function.
- Expansion: For node $S = \langle location, seen \rangle$, expansion of S generates the successors of S, which is the set of states reached by applying all legal movements from $S.locaton$. Each child S' of S corresponds to a given legal movement. $S'.location$ is the neighboring cell of $S.location$ derived from the movement. $S'.seen$ is first inherited from $S.seen$. Then we add to $S'.seen$ all the cells have seen before in $LOS(S'.location)$. All generated nodes will be pushed into the priority queue *OPEN* in according to the value of the priority evaluation function.

We use WRP-A* to denote the variant of A* that works on the search tree defined for WRP. The remainder of this chapter is based on Skyler et al 2022, and summarizes the key techniques including the admissible heuristics used for WRP-A* and some efficiency improvement measures.

2.3.1 Aggregating Singleton Heuristic

The first heuristic is based on the idea that, to solve WRP, the watchman agent must have visibility of each cell in $S.unseen$. For a given State S, the shortest paths between $S.location$ and any cells in $S.unseen$ indicates a possibly optimal path. Therefore, every cell $p \in S.unseen$ (we call it pivot) has their own shortest paths and we define its *Singleton heuristic* as the minimal distance from $S.location$ to a cell $q \in LOS(p)$.

$$h_p(S) = min_{q \in LOS(p)} distance(S.location, q) \tag{2.4}$$

where $distance(x, y)$ is the cost of the shortest path between cell x and y. For every pivot, the singleton heuristic $h_p(S)$ is admissible because in order to see all cells in grid-map, the watchman must travel to a cell with visibility to p. As $h_p(S)$ represents the minimum distance among all such cells, it ensures the admissibility of the heuristic.

Since only one heuristic value is needed for a given state S, we can take the maximum of the singleton heuristic among all pivots while maintaining admissibility [12]. To do this, we calculate all of the singleton heuristics and choose the maximum value. The *aggregating singleton heuristic*, denoting by h_{AGS} is defined as:

$$h_{AGS}(S) = max_{p \in S.unseen} h_p(S) \tag{2.5}$$

2.3.2 Graph Heuristics

Skyler et al also proposed two heuristics based on a graph called the *Disjoint LOS graph* $G_{DLS}(V, E)$. G_{DLS} is abstracted from the input grid-map $M(V, E)$ for every node S in *OPEN*. For clarity, we want to emphasize that the term *cell* denotes a cell of the grid and the term *vertex* denotes a vertex of G_{DLS}.

Two cells $x, y \in V$ are *LOS-disjoint* iff $LOS(x) \cap LOS(y) = \emptyset$. Meaning there is no common cell that has a LOS to both x and y. Similarly, a set of cells Ω is *LOS-disjoint* if it satisfies $\bigcap_{v \in \Omega} LOS(v) = \emptyset$.

We next define the way of building a G_{DLS} based on a grid M. Figure 2.1 presents an example with 4-way LOS. Figure 2.1(a) shows our grid-map which is the Root node. $G_{DLS}(S)$ shown in Figure 2.1(b) is built as follows.

Vertices of G_{DLS}: The set of vertices of $G_{DLS}(S)$ is created from a subset of cells from M as defined below. These vertices are classified into three types, each with a different color (Green, Red and Yellow):

- Watchman Cell (Green): The watchman cell (F) is associated with $S.location$
- Pivots (Red): The pivots vertices (A, C and G) are a *LOS-disjoint* set of cells from $S.unseen$. Note that there are multiple ways to determine the set of pivots. The method used in this context is described in Section 2.3.3.
- Watchers (Yellow): The Watchers (B, D, E, H, I and J) are all the cells that have been seen by each pivot from *pivots*. Every watcher is exactly seen by one pivot due to the *LOS-disjoint*.
- Other (Gray, White): Some cells in M do not have vertices in $G_{DLS}(S)$ and are therefore excluded. Grays cells are omitted because they are in $S.seen$ and white cells are excluded because they are failed to be selected as pivots although they are in $S.unseen$.

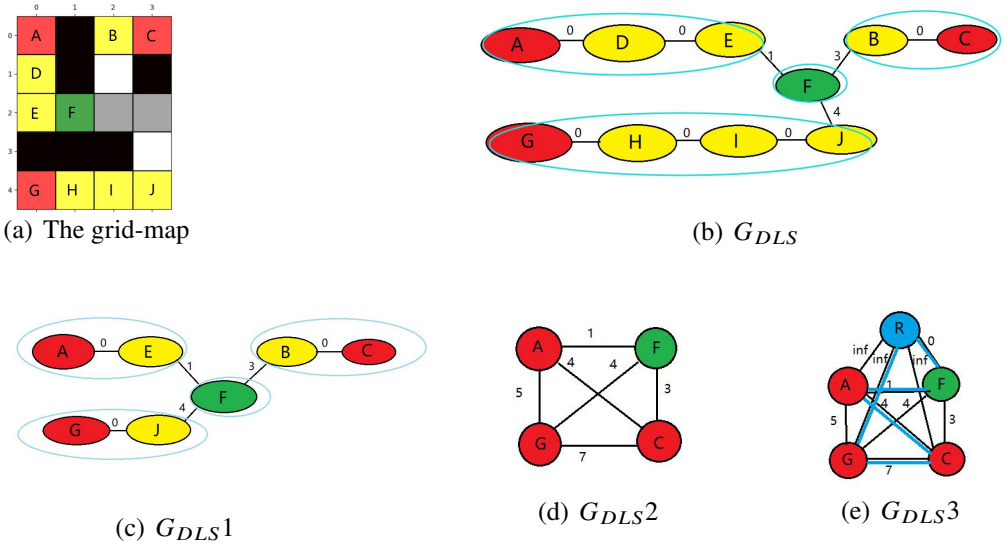
A pivot p and its watchers are denoted as *component p* of $G_{DLS}(S)$. Especially, watchman cell is a component too. There are four components with a blue circle in Figure 2.1(b) (component(A), component(C), component(F) and component(G)).
Edges of G_{DLS}: All edges $(x, y) \in E$ where both x and y exist in G_{DLS} are remained in G_{DLS}. After that, we find out all cells that were omitted from G_{DLS} and *contract away* their corresponding vertices [8]. Edges in G_{DLS} can be classified into two types. Their cost represents travelling cost for a watchman to see the pivots.

- Watching edges: Edges that only connect vertices belonging to the same component G_{DLS}. Such edges have a cost of 0 (e.g., c(A, D)=0).
- Traveling edges: Edges that connect vertices across components. The cost of these edges is the shortest distance between the two vertices (e.g., c(F, J)=4, c(B, F)=3).

Simplifying G_{DLS}: Since we are interested in finding a minimum pivot-covering path, we can simplify G_{DLS} by contracting away many of the vertices of G_{DLS} while preserving the fact that a minimum pivot-covering path is a lower bound on the remaining cost from node S. To achieve this, watchers are classified into two classes.

- Frontier watchers: Watchers that have edge connecting a vertex in another component in M.
- Internal watchers: Watchers(non-pivot) that have all their edges inside the component.

All Internal watchers are contracted away and only frontier watchers remain in G_{DLS}. We call the resulting graph $G_{DLS}1$ and is shown in Figure 2.1(c).



(a) The grid-map (b) G_{DLS}

(c) $G_{DLS}1$ (d) $G_{DLS}2$ (e) $G_{DLS}3$

Figure 2.1: An example of a grid and the derived G_{DLS}

2.3.2.1 MST Heuristic

A minimum spanning tree (MST) is a minimal connected subgraph of a connected, edge-weighted undirected graph, which connects all the vertices with the minimum total edge weight. The minimum spanning tree problem originated in the 1920s and many algorithms have been developed over the years to find MST efficiently. Among them were R. C. Prim and J. B. Kruskal, whose algorithms are very widely used today. The algorithms known as Prim's algorithm [23] and Kruskal's algorithm [15].

The Minimum Spanning Tree (MST) of a graph is the spanning tree with the minimal sum of edge costs. The MST heuristic $h_{MST}(S)$ computes a MST of $G_{DLS}1(S)$.

2.3.2.2 TSP Heuristic

The objective of the TSP is to find the path starting and finishing at the same vertex which visits every vertex exactly once with minimum sum of weight. TSP has been intensively studied and many heuristics and exact algorithms are known like tabu search [9], ant colony [4], etc.

Here our goal is to find a minimum pivot-covering path (i.e. a path that starts at watchman cell and passes through all the components) in G_{DLS}. To do this, we further abstract $G_{DLS}1$ to $G_{DLS}2$, which is a homomorphic abstraction of $G_{DLS}1$ in which all vertices within a component are merged into a single vertex. New vertices represent the corresponding components, as shown

in Figure 2.1(d). $G_{DLS}2$ is a complete graph (clique). The cost of any edges in $G_{DLS}2$ is the minimal cost among all edges that connect these components in $G_{DLS}1$, and all the edges have non-zero cost.

In TSP, a minimal cycle path (starting and finishing at the same vertex) is calculated. However, based on our goal, we expect the minimum-cost Hamiltonian path with a specific start vertex in a clique graph. Therefore, $G_{DLS}2$ is modified to a new graph $G_{DLS}3$ so that TSP solvers can be applied. A single reference vertex R is added to $G_{DLS}3$. It is connected by edges to all other vertices in the graph. The cost of the edge connecting R to the watchman cell F is 0 while cost of others edges are inf, a constant that larger than the sum of all edges in $G_{DLS}2$. $G_{DLS}3$ is shown in Figure 2.1(e).

Observing that a TSP answer tour for $G_{DLS}3$ (R-F-A-C-G-R in our example), to find a minimum path, the edge between R and F must be chosen. In other words, the vertex representing watchman (i.e F) must be included in the tour. If we remove R and its two incident edges from tour then the minimal-cost Hamiltonian path (F-A-C-G) is left, F is specified as our start vertex.

2.3.3 Pivot Selection

In order to produce high heuristic values, it is desirable to have a maximal size for the set of pivots. A set L is maximal if no other cell is *LOS-disjoint* to any of cells of L. The pivots are chosen using the following greedy strategy: a cell c is selected as pivot if c is *LOS-disjoint* to any cells of L and has the smallest visibility range ($|LOS(c)|$) among all empty cells except cells in L (=$V - L$). In others words, the pivots are continually selected in increasing order of their visibility range.

2.3.4 Comparison of WRP Heuristics

Skyler et al compared the results of search using various heuristics in WRP. Among the graph heuristics, the value of h_{TSP} tends to be larger than the value of h_{MST} due to the different objectives of these heuristics. This means that h_{TSP} is more informed, but it also comes at the cost of being computationally more expensive as TSP is a NP-hard problem. There is therefore a trade-off between the accuracy of the heuristic and the time required to compute it. Skyler also found that while the TSP heuristic is generally superior, there are cases where AGS should be given priority. AGS outperforms the graph heuristics when the solution path contains a very small number of jumps, which typically occurs when the map owns a large open area. In such maps, when building a G_{DLS} for node u, each cell in the map will have many watchers, resulting in a low cost for the pivot-covering path. In this situation, AGS performs better than the graph heuristics both in terms of node expansion (it's important to note that WRP and MWRP have

different definitions of node expansion) and computational time. AGS is known for its fast computation, as it does not require building a time-consuming G_{DLS}. In some cases, AGS can be up to 30 times faster than the graph heuristics. This suggests that a simple, fast heuristic can outperform a more complex heuristic. We aim to confirm this conclusion in a later experiment.

2.3.5 Jump to Frontier (JF)

Normally, node generation involves performing all legal movements from the current location (referred to as $S.location$). However, as shown in $G_{DLS}1$ (Figure 2.1(c)), the size of the search tree can be significantly reduced by considering Jumping to Frontier.

Given a node S and its corresponding $G_{DLS}1$, the watchman path π want to go further from $S.location$ will include at least one watcher (yellow) for each component, and path will first meet what we call *frontier watchers* as they are neighbouring to $S.location$. Thus the optimal watchman path will at least include one of them. Therefore, We can choose *frontier watchers* as choices for our next step instead of legal movements.

To ensure that the search process is optimal, it is necessary to add all frontier watchers as neighbors of S in the search tree. In other words, when expanding node S, we generate one child C for each edge in $G_{DLS}1(S)$ that connects the watchmen cell to a frontier watcher. The path cost of node C is defined as $cost(C.path) = cost(S.path)+cost(\pi(WatchmanCell, FrontierCell))$, where $\pi(WatchmanCell, FrontierCell)$ represents the shortest path between the frontier cell and the watchman cell in grid-map.

Note that in Figure 2.1(a), white cells and grey cells are ignored. Grey cells are trivial because they are in $S.seen$, but it is possible that the optimal watchman path passes through a white cell. To account for this possibility, we iterate all white cells. For a given white cell called w, we add it as a pivot and all other white cells in $LOS(w)$ as watchers. Then we build a temporary component for them and add it into $G_{DLS}1$. These temporary components are only used for node generation and are not considered by the graph heuristics. By using the technique called Jump to Front (JF), we can perform the search process with excellent speed

2.3.6 Sub-optimal Variants of WRP-A*

It is common that for a NP-hard problem, a trade-off between solution quality and running time is needed. In this section, we briefly introduce some measures that can significantly accelerate the search process.

WA*: Using Weighted A* (WA*) [22] instead of A* is a common speeding up measure. WA* is a best-first search algorithm, in which f is modified to place additional weight on the heuristic function h, i.e. $f'(u) = g(u) + w * h(u)$ where $w \geq 1$. The solution returned by WA*

is guaranteed to be $w \times C^*$ (w-admissible), where C^* is the optimal cost.

Ignoring White cells (IW): Recall that in Jump to Frontier(JF), we add white cells and their LOS as temporary components to maintain optimality. However, we propose to ignore these white cells in order to reduce the size of the search tree. This will potentially result in a non-optimal path, but as demonstrated by Tamir Yaffe et.al [26], this will only occur in rare cases.

Removing Weakly-Redundant Components (WR): When solving larger map instances, the number of pivots will increase as we as are be able to identify more *LOS-disjoint* pivots for G_{DLS}. Assmuming that JF is applied, more pivots can result in the increment of the search tree size, as pivots may be add as children of a node in the search tree which means more path options. To reduce the size of G_{DLS} and minimize the impact on the solution quality, we define what we call *weakly-redundant* components to identify which component should be removed. Let Q be a component with q as its pivot. A component P is *weakly-redundant* with respect to Q if one shortest paths from $S.location$ to q passes through component P. Such a component can then be removed from G_{DLS} and will not be used for node generation and graph heuristic computation.

Chapter 3

Multiple Watchman Route Problem

3.1 Problem Definition of MWRP

As MWRP is a multiple watchmen case of WRP, we keep some expressions that are used in WRP like LOS.

The input of MWRP is the tuple $\langle M, k, starts, LOS \rangle$, where $M(V, E)$ is a grid-map. k is the number of watchmen. $starts = \{s_i \in V, i = 1, 2, ..., k\}$ denotes the start location of each watchman. The LOS function can be any arbitrary function for each watchman. Each of watchmen can perform a move action to one of its empty neighboring cells per time step. The task is to find a path for each of the watchmen from its start cell through the grid such that all empty cells in the map will be covered by line-of-sight (LOS) from at least one cell of one of the paths [18]. There are two objective functions are considered: (1) MinMax: Minimize the cost of the longest path among the paths. (2) MinSum: Minimize the sum of the costs of all the paths.

3.2 Previous Work on MWRP

Most of researchers are considering solving MWRP in a polygon instead of a grid-map before. To our best knowledge here are mainly two methods to solve MWRP: AGP+MTSP and SOM. We explain the keys of these methods below:

3.2.1 AGP+MTSP Method

A decouple method for solving MWRP is widely used. MWRP is divided into two problems: Art Gallery Problem (AGP) and Multiple Travelling Salesman Problem (MTSP). MTSP is a multiple salesmen case of TSP, similar with the relationship between WRP and MWRP. WRP can be considered as a variant of the well known AGP proposed by Victor Klee in 1973. AGP is

inspired by a question:"What is the smallest number of guards needed to guard an art gallery?" The idea behind AGP is to place the minimum number of guards necessary to collectively cover the interior of a given simple polygon. The guards in AGP are static and the goal is to minimize their number. Once the guards have been identified, Ths task is to connect them with k paths in an optimal way. The problem can be formulated as MTSP and these k paths (considered as k watchmen routes) represent a solution to MWRP.

AGP: In order to improve upon theoretically-induced results and to deal with d-visibility problem where the watchman's line of sight is restricted to a maximum distance of d. several algorithms have been developed to find approximate solutions to AGP. Kazazakis et al [13] developed a sensor placement algorithm to solve AGP. This algorithm assigns guards with d-visibility and is based on the deterministic decomposition of a polygon W into a set of convex polygons. Each convex polygon is guarded by one guard, and in order to meet the d-visibility requirement, the distance between the guard and a vertex of the guarded polygon must be less than d. The primal convex partition is found by Seidel's algorithm [24] and if a convex polygon is too large, it is divided into convex sub-polygons, until each sub-polygon can be covered by one guard with d-visibility. The similar algorithms are also used by [3] [10].

MTSP: The solution to MWRP is found by solving MTSP on a graph $G(V,E)$, where V represents the identified guards and E is a set of edges with costs determined by the shortest path between guards. Numerous methods have been proposed to find optimal solutions to MTSP, including Ant Colony Optimization, Genetic Algorithm, Simulated Annealing, and various neural network approaches. In particular, the approaches that based on Self-Organizing Map (SOM) are very promising [27] [16].

A SOM [14] is an unsupervised machine learning technique used to produce a low-dimensional representation of a higher dimensional data set while preserving the topological structure of the data. In the context of path finding, we first train SOM on a set of input data that represents the area we want to find a path through. Such as feeding the SOM a series of coordinates of cities. Once the SOM is trained, it can used to find the shortest path.

We first provide a detailed overview of the schema of SOM in the context of TSP and then extend this schema to MTSP form. The used two-layer competitive learning network consists of two dimensional input vectors and an array of output nodes. An input vector i represents coordinates (g_{i1}, g_{i2}) of the guard g_i and weights w_{j1}, w_{j2} between input and any nodes j are interpreted as coordinates of node j. Assume the number of guards is m and the number of nodes is n. It is worth noting that n is typically much more than m in order to form a path. We iteratively update the weights of the connections between the input vector and the nodes by inputting one guard's coordinates once a time. For a given guard, the output nodes compete to be the winner node and its neighbouring nodes are updated in order to get closer to the presented

guard according to one neighbouring function. The presentation of all guards to the network is repeated until all guards have their winner nodes closer than one given threshold. A guard tour (watchman path) can be found by connecting nodes together to form a ring (every node j have its w_{j1}, w_{j2} that means its coordinates). The final length of guard tour is computed of guard-guard distances.

In MTSP, the goal is to find a tour for each of k salesmen (watchmen), To do this, an individual ring of nodes is created for each watchman. Remember that the number of guards m and the number of nodes n, k chains can be formed such that each consists of $n = \frac{2m}{k}$. Similar to TSP, the algorithm represents the path of each particular watchman by a chain of nodes, where neighboring nodes are connected. Initially, the nodes on the chain are positioned on a small ring close to the starting location for each watchman. At each iteration, the nearest node to a randomly selected guard is determined and together with its neighbors moved closer to the guard. Eventually, the k chains of nodes (k watchmen paths) will be the solution to MWRP.

3.2.2 SOM Method

SOM can be also used to address MWRP directly, without dividing it into two subproblems. A SOM based adapatation procedure is developed to address MWRP with a d-visibility range in the polygonal domain W by J Faigl [5]. The algorithm is based on the adaptation schema for MTSP with the MinMax criterion. In this case, we will only introduce the SOM schema in the context of the WRP, as the extension of the proposed WRP algorithm to MWRP is analogous to the extension of the TSP algorithm to MTSP.

Given a polygon map W, WRP involves finding a route that passes through a subset of convex polygons from the cover set such that the union of these polygons covers the entire map W. The cover set consists of all possible convex polygons of W and is determined by breaking the map into a series of triangular meshes using a mesh generator. Any combination of these meshes can be used to form a convex polygon, as long as the distance between any two points on the polygon is less than d (d-visibility).

To find a watchman path, problems below need to be addressed:

1. calculate the current path coverage and uncovered parts of W
2. find out the attraction points of the uncovered parts
3. attract nodes towards attractions points

For problem 1, the coverage of path is determined according to the following process: 1) For a given sequence of current path $(p_1, p_2, ..., p_n)$, we determine the set of incident triangles

by identifying any triangles that are on the path. 2) To calculate the coverage, we also need to find the incident polygons. Since convex polygons are composed of multiple triangles, we define a related set of polygons as the union of all polygons P_i that contain any of the triangles T_i in T_r (i.e. $P = \bigcup_{T_i \in T_r} \{P_i | T_i \in P_i\}$). 3) The coverage of the path is then found as the union of all triangles T associated with all incident convex polygons P_i in P. (i.e. $\bigcup_{P_i \in P} \{T | T \in P_i\}$).

For problem 2, once we have the coverage of the current path, we want to expand this coverage to the entire map W. To do this, we can use the centroid of each uncovered triangle as an attraction point.

For problem 3, unlike like TSP, our aim is to find a route taht allows the watchman to "see" all triangles rather than necessarily visiting each one. Therefore, the update rule has been modified to prevent nodes from being placed unnecessarily close to the attraction point. The update process is otherwise similar to the one described in Section 3.2.1 for TSP.

3.2.3 Disadvantages of Previous Work

Although both existing methods can solve MWRP in a polynomial domain, the d visibility based algorithms determine that there is a restrictions on the use of LOS functions. These methods are unable to handle MWRP with a free LOS function setting, such as different LOS function applications for different watchmen. In addition, the solutions produced by these methods must be cyclic, so they cannot choose any specific cell as the goal. Additionally, these methods have their own disadvantages.

The main shortcoming of the AGP+MTSP method is that the combined objective is inconsistent with that of MWRP. In AGP, the aim is to find a set of guards that can statically cover the entire map, ignoring cells that may be visible during movement along the path. Therefore, even if the MTSP solver provides an optimal solution for connecting these guards, there is no guarantee that the overall solution will be optimal for MWRP.

As for the SOM method, the training process is designed to find a circular minimum path instead of stopping anywhere that satisfies the termination conditions. As a result, the solution tends to always be a circle.

In our research, we aim to remove the limitations of both approaches (forming a circular path and limited LOS function choices) while also improving performance.

Chapter 4

Proposed Method

Algorithm 1 Main Framework

Input: grid-map M, number of watchmen(cluster) k, LOS function $LOSType$
Output: k best watchmen paths $\{\pi_i^* = \langle p_1^{(i)} = s_1, p_2^{(i)}, ..., p_n^{(i)} \rangle, i = 1, 2, ..., k\}$

Initialization: cells set $V = \{c_i | c_i \; is \; not \; obstacle\}$, priority queue $Q \leftarrow []$, empty state S, edges set E, the max gap can be ignored ϵ, $values \leftarrow []$
$starts \leftarrow$ RandomChooseK(V)
$d, APSP, LOS \leftarrow Preprocessing(LOSType)$
$ClustersSet \leftarrow MultiClassKmeans(k, starts, distance, V)$
$S.clusters \leftarrow ClustersSet$
$S.starts \leftarrow starts$
for cls in $ClustersSet$ **do**
    $path_{cls} \leftarrow$CalcNewPath(cls, s_{cls})
end for
$S.paths \leftarrow \{path_1, path_2, ..., path_k\}$
$S.value \leftarrow$EvalueFunc($S.paths$)
$S^* \leftarrow S$
PUSH(Q, S) and PUSH ($values$,$S.value$)
while IsNotImproved($values, S^*.value$) **do**
    if IsEmpty(Q) **then**
        break
    end if
    $S' \leftarrow$POP(Q)
    PUSH($values$,$S'.value$)
    $S^* \leftarrow$Min($S^*.value$,$S'.value$)
    $IterativeRepairingSearch(E, S', APSP, \epsilon)$
end while
Return $S^*.paths$

In this section, we will describe the details of the main framework of our algorithm which is shown in Algorithm 1. we try to solve MWRP on a grid map using a clustering algorithm and

generate-and-repair search algorithm.

In the first stage, we classify all cells in the grid into different groups using a multi-class clustering algorithm. We then apply a generate-and-repair search algorithm. At the route generation level, we find the k best paths at k clusters corresponding to the watchmen using the WRP-A* solver. At the route repair level, we attempt to minimize the LOS duplication between different watchmen paths during the search process. We use an evaluation function to score the solution and record the state S^* with the current best value. We continually generate new nodes through reassigning cells and recalculating the path at the route repair level, expanding from the current best node. This process is repeated until the value of S^* no longer improves.

To solve MWRP with our algorithm, we treat it as a search problem and define the following concepts:

- State: A state is tuple $\langle clusters, starts, paths, value \rangle$, *clusters* indicates the clustering result of each cell in the map and *paths* is a set of paths for each watchman corresponding with clusters. *value* is calculated by the evaluation function, representing the solution quality of the current state.

- Path and Cost: the path π in state S is calculated by the WRP-A*, based on the cells of the corresponding cluster and start location. The cost is the sum of the traveling cost of path π.

- A solution: k watchmen paths $\{\pi_i = \langle p_1^{(i)} = s_1, p_2^{(i)}, ..., p_n^{(i)} \rangle, i = 1, 2, ..., k\}$, $P = \bigcup_i \pi_i$ and $\bigcup_{c \in P} LOS(c) = V$

- Node: A node refers to a state in the MWRP search space.

- Root: *Root* is a node with an initial solution first returned by WRP-A* at route generation level. In *Root*, all cells are assigned to a cluster and paths are determined such that we build a search tree for a better solution.

- Goal: Any node when the best value of node is converged. Therefore we don't specify the ending cell of path and the assigning way of clusters.

- Generation: We first get *Root* after applying the WRP-A* at route generation level and we generate nodes and add them to priority queue Q at route repair level with an iterative repairing search algorithm which is mentioned below.

- Expansion: Every time iterative repairing search algorithm ends, we will take the first node from the priority queue Q for expansion (greedy strategy).

4.1 Preprocessing

To avoid wasting computational resources on repeated calculations, we first calculate the following functions in a preprocessing phase, which can then be used as a lookup table in the main algorithm.
x, y representing two cells from the input grid map and V is the cells set.

- $distance(x, y)$: the cost of shortest path between x and y
- $APSP(x, y)$: the shortest path between x and y.
- $LOS(c)$: all cells can see from $c \in V$

The number of cells is denoted by $n = |V|$. The time complexity for building the d table and $APSP$ table is up to $O(n)^3$ (built by Floyd-Warshall algorithm [7]). The time taken to build the LOS tables is $O(p \cdot n^2)$, where p is the time required to compute whether there is a LOS between two cells. These lookup tables can be fully built in a preprocessing phase or lazily during the search process with no significant difference in time. While it takes polynomial time to build these tables, the main problem is NP-hard. Therefore, the time and memory required to build these tables is negligible compared to the resources needed to solve the main problem.

4.2 1st Stage: Clustering Algorithm for MWRP

This stage aims to create a set of clusters Ω with size k so that we can partition the entire grid map to k different groups for later pathfinding. There are two additional requirements for this clustering algorithm: 1) each cluster must be connected, meaning that every two distinct nodes in the cluster must be joined by a path formed of edges in E. 2) The size of each cluster should be balanced, meaning that we aim for each cluster $C \in \Omega$ to have a similar size keep the size of each cluster balanced, which means we hope each cluster $C \in \Omega$ contains similar size.

The multi-class k-means algorithm performs well in satisfying the balance and connectivity requirements, even without a separate phase for improving balance. The pseudocode for the multi-class k-means algorithm is provided in Algorithm 2 below.

the detailed workflow of the algorithm can be summarized in the following steps:

1. We initialize k cluster's centroids as each watchman's start position and define $dc(c)$: the distance from the centroid r of cluster C to cell c if c is belong to Cluster C. If c does not belong to any clusters, let $dc(c)$ be undefined.

Algorithm 2 $MultiClassKmeans(k, starts, distance, V)$

Input: number of watchmen(clusters) k, $starts = \{s_1, s_2, ..., s_k\}$, $distance$, cells set V
Output: Set of clusters $\Omega = \{cls_1, cls_2, ..., cls_k\}$

Initialization: for $i \in \{1, 2, ..., k\}$, $cls_i.centroid \leftarrow s_1$
$\forall c \in V, dc(c)$ undefined
repeat
  while $dc(c)$ undefined for any $c \in V$ **do**
    for $i = 1$ **to** k **do**
      $candidates \leftarrow cls_i.reachable$
      /*$reachable$ sort in ascending order by distance from centroid */
      for $cand$ in $candidates$ **do**
        $close\_d \leftarrow distance(c, cls_i.centroid)$
        if $dc(c)$ has defined and $close\_d > dc(c)$ **then**
          continue
        else
          $dc(c) \leftarrow close\_d$
          $cls_i.cells \leftarrow cls_i.cells \cup c$
        end if
      end for
    end for
  end while
  for $i = 1$ **to** k **do**
    $cls_i.centroid \leftarrow$Mean$(cls_i.cells)$
    $cls_i.cells \leftarrow \emptyset$
  end for
until all centroids converge

2. For keeping connectivity, we maintain a list *reachable* in each cluster for identifying all the cells that can reach from a cluster and sort them in ascending order based on distance from centroid.

   Every time we perform a cell assignment, we choose the closest cell in the *reachable* set for the cluster. If a cell c has been already assigned to another cluster, we must consider whether it should belong to multiple clusters. If the current closest distance is greater than $distance(c)$, it means that cell c has belonged to a closer cluster, so we stop the assignment. Otherwise, we treat c as a multi-class cell and add it to the current cluster.

3. after all cells are assigned, we recalculate the new centroid as the mean of each cluster.

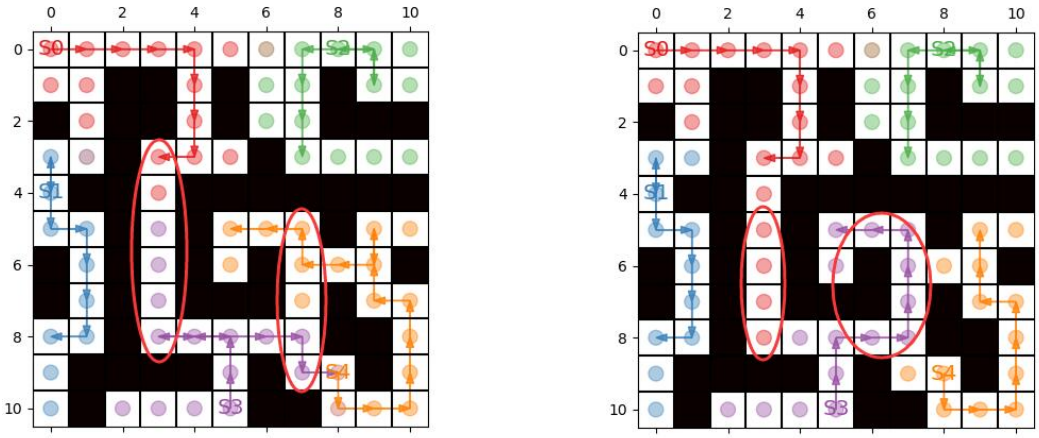
4. repeat steps 2 and 3 until all centroids converge.

4.3 2nd Stage: Generate-and-Repair Search Algorithm

Using a clustering algorithm, we have already obtained a set of clusters Ω. We then use a generate-and-repair search algorithm to search for the best solution to MWRP based on this partitioning.

Route generation Level: We partition the entire grid map into k areas based on the clustering set and treat each area as a Watchman Route Problem. We then use the existing sub-optimal WRP-A* solver to independently find the k best paths, creating a node called *Root*. However, the locally optimal solution may not lead to a good global solution. In most cases, a broad LOS function means that even though each cell only needs to be seen once, the local solution may result in many cells being seen repeatedly. We call it LOS duplication and Figure 4.1 represents an example. *Sn* denotes the start location of watchmen and the arrow shows their paths. Different colors denote the clustering result and the color of multi-cluster cells will be determined by the RGB average value of the color represented by cluster (e.g., the cell in the 0-th row and the 6-th column, red and green blend to turn brown). It is obvious that LOS duplication (4-way-LOS) occurs where cells are circled, hence there are some redundant movements for watchmen. Figure 4.1(b) shows a better solution after removing LOS duplication by reassigning cells.

Route repair Level: We try to minimize the LOS duplication problem by reassigning cells among clusters to find a new path during the search process. We use the evaluation function $f(n)$ to evaluate the quality of the solution. $f(n)$ consists of three parts: the variance of path length, the maximum length, and the average length. In some extreme cases, the variance may be much larger than the other two parts. To balance the weight of each value, we add a scaling factor w to variance, that is $w = \frac{variance(n)}{MAX\_VARIANCE}$. This function helps us to find a solution that



(a) Solution with LOS duplication (b) Better solution

Figure 4.1: An example of LOS duplication

simultaneously minimizes the average and variance with a smaller value indicating a higher quality solution.

$$f(n) = w * variance(n) + max(n) + average(n) \tag{4.1}$$

the detailed workflow of the algorithm can be summarized in the following steps:

First, traverse all edges $\in E$ to find the edge (x, y) at the boundary of the cluster, meaning that both ends of the edge belong to different clusters. Let's assume x belongs to *clsA* and y belongs to *clsB*. We then consider reassigning cells from *clsA* to *clsB* (since edges in E are undirected edges, we only need to consider one direction assignment). Otherwise, we set a gap ϵ to ensure that the path length of *clsA* is longer than *clsB* and that their length difference is greater than ϵ. The reason is that, the new path length of a cluster that loses cells will always be less than or equal to the original length, while the other cluster's length will increase or keep the same. To balance the path lengths of each cluster and minimize the value of evaluation function, we include this conditional statement (line 6 of Algorithm 3).

Second, if we only assign cell x from *clsA* to *clsB* at a time, it has a trivial help for proceeding the search process. Suppose that if cell x is not on the path of *clsA*, even if it has been assigned to *clsB*, it is likely that the paths on both clusters may not be changed at all. To enhance the pace of path repairing process, let z be the cell on the path of *clsA* nearest to x, we try to assign x, z, the cells on the shortest path between x and z and other cells related to z (lines 9 to 12 of Algorithm 3).

We define the RelativeCells(z), the set of cells that need to be reassigned if z is reassigned. If removing z from *clsA* doesn't influence the connectivity of *clsA*, then the RelativeCells(z) = z. For example, suppose s is the successor of z. If there is another cell that can be the new predecessor of s while maintaining connectivity, we can keep all subsequent cells of z.

Otherwise: 1) if z occurs only once in $Path_A$, to maintain connectedness, RelativeCells(z) = z + all cells on $Path_A$ following z. 2) if z occurs multiple times on $Path_A$. Let z_0 and z_1 be the two instances of z which are furthest apart in $Path_A$. Then RelativeCells(z) is z_0, z_1, and the subpath of $Path(A)$ between z_0 and z_1.

The set of cells that are reassigned is $need\_assign$, which includes x, z, APSP(x, z), and RelativeCells(z).

In addition to cells in $Path_A$, we should focus on cells that become unreachable from $clsA$ if $need\_assign$ is reassigned. we know that every cell in a cluster must have at least one inner path (the cells of path are from the same cluster) to the centroid. We define the set of outlier cells as the cells in *clsA* that are unreachable from $cls.centroid$ (i.e. no inner path π between outliers and $cls.centroid$) and the definition of reachable is $Reachable(start, cells) = \{c | \pi(start, c) \subset cells\}$. Therefore, after assigning the cells in $need\_assign$, we need to check which cells will be outliers and assign them together (lines 14 to 16 of Algorithm 3).

After the assignment is done, we define $CalcNewPath(cls.cells, start_{cls})$, recalculate the new path for *clsA* and *clsB* with WRP solver. With other clusters that have no changes, creating a new state S' and push it into the priority queue.

Remind that in a multi-class scene, there are some cells belonging to multiple clusters, but actually it has no impact on our search process. we can treat it as a single cluster situation. For example, one end of edge belong to *red, green*, another end belong to *blue, green*, then we just consider the assignment of *red to blue, red to green, green to blue, green to green*.

4.4 Efficiency Improvements for MWRP

Even though our algorithm can theoretically find a solution to MWRP, it is a NP-hard problem, so it is likely that we will not be able to get an answer quickly for more complex instances. To speed up the search process while minimizing the performance loss of the solver, we propose two methods. Algorithm 4 shows an improved version of the entire process.

4.4.1 Hash Table Structure

We use a hash table structure for two purposes: 1) Storage of existing paths for reuse. 2) Skipping nodes that have already occurred. At route repair level, we continually generate new nodes by iterative repairing search and expand the current best one until no improvements. The most time-consuming part is that we have to apply WRP-A* twice to get new paths for watchmen whose cells have changed.

For the first purpose, Since we calculate the path in the unit of cluster, the path of the

Algorithm 3 $IterativeRepairingSearch(E, APSP, S, \epsilon)$

Input: edges set E, $APSP$, current state S, the max gap can be ignored ϵ, priority queue Q

Initialization: $need\_assign \leftarrow \emptyset$
$S' \leftarrow$ Copy(S)
for edge (x, y) in E **do**
  $clsA \leftarrow S'$.Cluster(x), $clsB \leftarrow S'$.Cluster(y)
  /* considering assign cells of $clsA$ to $clsB$ */
  if $clsA == clsB$ or len(path$_{clsA}$)-len(path$_{clsB}$)$< \epsilon$ **then**
    continue
  end if
  $z \leftarrow$ the cells on path$_{clsA}$ nearest to x
  $need\_assign \leftarrow x \cup z \cup APSP(x, z)$
  if CheckConnectivity($need\_assign$) is False **then**
    $need_assign \leftarrow need_assign \cup$RelativeCells($z$)
  end if
  $leftcells \leftarrow clsA.cells \setminus need\_assign$
  $outliers \leftarrow leftcells \setminus$Reachable($clsA.centroid, leftcells$)
  $need\_assign \leftarrow need\_assign \cup outliers$
  $clsA.cells \leftarrow clsA.cells \setminus need\_assign$
  $clsB.cells \leftarrow clsB.cells \cup need\_assign$
  $path_{clsA} \leftarrow$CalcNewPath($clsA.cells$, $start_{clsA}$)
  $path_{clsB} \leftarrow$CalcNewPath($clsB.cells$, $start_{clsB}$)
  PUSH(Q, S')
end for

watchman with the same start location and cells of cluster will be the same. Thus there is a great possibility that the same path repeatedly occurs during the iterative repairing search process and leads to the waste of computing resources. To prevent the waste, we create a hash table $paths\_table$, making use of the hash algorithm to create a unique key $path\_code$ based on the cells and start location of watchman, let the path of watchman be the value. We can lazily build $paths\_table$ during search process.

For the second purpose, we aim to prevent the same assignment from occurring multiple times in a single iteration. To do this, we traverse all edges to find cells that meet the assigning requirements. For any cell x, it should only happen once that the x and its related cells=$x \cup z \cup path(x, z)$ are assigned from *clsA* to *clsB* in an iteration. However, in practice, any cell x has at least two edges (e.g., the cell located in the upper left corner) and if more than one edge is connected to the same cluster, it can result in the same assignment being performed multiple times. To avoid this, we use a hash table H and create a unique $hash\_code$ based on $x, clsA, clsB$ using a hash algorithm. With $hash\_code$, we ensure that the same assignment will only happen once.

4.4.2 Pre-pruning

As mentioned previously, we spend a large amount of time generating nodes. Although the hash table structure helps to reduce some repeated calculations, there are nodes with poor quality solutions that may not be removed from the priority queue until the value of S^* converged, leading to a waste of computing and space resources. To address this issue, we implement pre-pruning to prevent the generation of unexpected nodes.

We define the set size $len_{clsA} = |need\_assign \cap cells\ of\ path_{clsA}|$ and $len_{clsB} = |need\_assign \setminus cells\ of\ clsB|$ and assess len_{clsA} is the length that $paht_{clsA}$ will reduce and len_{clsB} is the length that $path_{clsB}$ will increase after assignment. We then can calculate $post\_value$ by evaluation function $f(n)$ as an assumption and compare $post\_value$ with $pre\_value = S.value$ which is the value of state before assigning. Based on this, we can easily evaluate whether an unbalanced result will be obtained after the assignment operation is executed and prevent it from occurring. Here, we use a multiple factor ε as a bound. If the value of $\frac{post\_value}{pre\_value}$ exceeds the bound, indicating that the quality of solution is not good enough to be generated.

Algorithm 4 $IterativeRepairingSearchV2(E, paths\_table, S, APSP, \epsilon_1, \epsilon_2)$

Input: edges set E, hash table $paths\_table$, current state S, $APSP$, the max gap can be ignored ϵ_1 and ϵ_2, priority queue Q

Initialization: $need\_assign \leftarrow \emptyset$, hash table $H \leftarrow \emptyset$
$S' \leftarrow$ Copy(S)
for edge (x, y) in E **do**
  $clsA \leftarrow S'$.Cluster(x), $clsB \leftarrow S'$.Cluster(y)
  /* considering assign cells of $clsA$ to $clsB$ */
  $hash_code \leftarrow$GetHash($x, clsA, clsB$)
  if $hash\_code$ in H or $clsA == clsB$ or len(path$_{clsA}$)-len(path$_{clsB}$)$< \epsilon_1$ **then**
    continue
  end if
  $H \leftarrow H \cup hash\_code$
  $z \leftarrow$ the cells on path$_{clsA}$ closest to x
  $need\_assign \leftarrow x \cup z \cup APSP(x, z)$
  if CheckConnectivity($need\_assign$) is False **then**
    $need_assign \leftarrow need_assign \cup$RelativeCells($z$)
  end if
  $pre\_value \leftarrow S.value$
  $post_value \leftarrow$CalcValue($need\_assign, clsA, clsB$)
  if $\frac{post\_value}{pre\_value} \geq \epsilon_2$ **then**
    continue
  end if
  $leftcells \leftarrow clsA.cells \setminus need\_assign$
  $outliers \leftarrow leftcells \setminus$Reachable($clsA.centroid, leftcells$)
  $need\_assign \leftarrow need\_assign \cup outliers$
  $clsA.cells \leftarrow clsA.cells \setminus need\_assign$
  $clsB.cells \leftarrow clsB.cells \cup need\_assign$
  for cls in $\{clsA, clsB\}$ **do**
    $path\_code \leftarrow$ GetHash($start_{cls}, cls.cells$)
    if $path\_code$ in $paths\_table$ **then**
      $path_{cls} \leftarrow paths\_table[path\_code]$
    else
      $path_{cls} \leftarrow$CalcNewPath($cls.cells$, $start_{cls}$)
      $paths\_table \leftarrow paths\_table \cup (path\_code, path_{cls})$
    end if
  end for
  PUSH(Q, S')
end for

Chapter 5

Experiment Results

We have implemented our framework on a PC with Microsoft Windows 11 system using Python 3.11.0 (released on Oct 24 2022). The tests were performed on a Microsoft Windows XP workstation on an 11th Gen Intel(R) Core(TM) i7-11800H 2.30 GHz CPU with 16GB of RAM. We performed extensive experiments on various types of maps using variants of MWRP that employed different heuristics and WRP-A* solvers (optimal and suboptimal) with varying numbers of watchmen. We report our findings here on representative example maps and some similar trends were observed across different maps. We showed our proposed efficiency measures significantly improves in saving time with a trivial loss in solution quality. The experiments demonstrate that our algorithm performs nearly as well as the baseline.

Here we introduce the definition of baseline, since MWRP is hard even to approximate, it can be challenging to accurately evaluate the results by comparing it to optimal solutions or even solutions that approximate the optimum. Instead, we apply the optimal WRP-A* to the same map to obtain the optimal solution (path) p^*. As a reminder, k represents the number of watchmen. To establish a baseline for our MWRP solver, we consider dividing the optimal solution path p^* into k segments, resulting in a baseline of $\frac{p^*}{k}$. However, the optimal WRP-A* may not perform well due to computing resource limitations when applied to larger maps. Therefore, for larger map experiments, we use a sub-optimal version of the WRP-A* algorithm to obtain our baseline. We also use this same setting for the WRP-A* solver applied in our route generation level to maintain consistency and demonstrate the great performance of our iterative repair algorithm. Both the optimal and sub-optimal versions of the WRP-A* solver were proposed by Skyler et al [26] and have shown strong performance on various maps.

Although our algorithm can adapt any kind of LOS function, we only use 4-way LOS during experiments for the sake of simplicity and clarity of results. In our experiments, we consider the following variants of our algorithm:

- algorithm with different heuristic function: Aggregating Singleton(AGS), MST, TSP

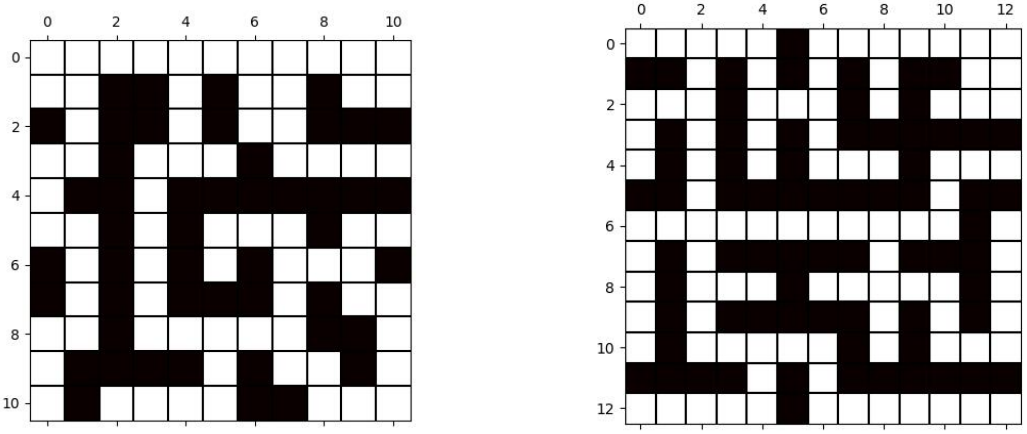
- algorithm with proposed efficiency improvements: hash table structure, pre-pruning,
- algorithm with speed-up techniques of sub-optimal WRP-A*: WA*, Ignoring White (IW), Weakly Redundant (WR)

5.1 Evaluation with Optimal WRP-A* solvers

5.1.1 Exploration of Proposed Efficiency Improvements

We conducted experiments on an 11x11 grid-map, as shown in Figure 5.1(a). With a relatively small map, we are able to provide a full comparison of all factors by optimal WRP-A* solver (meaning that we did not consider of WA*, IW and WR). For 2-5 watchmen, we generated 30 map instances independently by randomly selecting the start location of the watchmen.

In Table 5.1, we give the experiment settings, and the results are shown in Table 5.2. The columns represent the number of watchmen, hash table option, pre-pruning option (True indicates pre-pruning was used during search and False indicates otherwise), the number of expansions, the CPU time (in seconds), the cost of MinMax approach and the baseline for the MinMax approach. The rows correspond to the different combinations of watchmen, hash table and pre-pruning. It is clear that both proposed efficiency improvements significantly reduce the search time and the number of expansions. In the meantime, the effect on the quality of the solution (MinMax) is trivial. Especially when the number of watchmen is 2, the use of a hash table and pre-pruning reduces the number of expansions from 115.8 to 22.4 (nearly 81%) and the time from 325.1 to 0.2 (nearly 99%). It is hard to say which measure performs better, but applying both measures always leads to further reductions in time and expansions. Therefore, it is best to use both measures.



(a) 11 x 11 grid-map

(b) 13 x 13 grid-map

Figure 5.1: Small maps for experiment

5.1.2 Comparison with Baseline on Small Maps

In this section, we present a comparison of the results on two small maps (Figure 5.1) using the baseline. These two maps have similar characteristics, including a lack of large open areas and a high number of obstacles (11x11: 48 of 121, 13x13: 71 of 169). Based on our findings in Section 2.3.4, we expect graph heuristic functions to perform better than AGS in these types of maps, as G_{DLS} is likely to be more informative. To further confirm these results, we generated 50 map instances with randomly selecting start locations for the watchmen. The experimental setting are given in Table 5.3. We present a full comparison result of different watchmen and heuristic functions of WRP-A* and the results are shown in Table 5.4.

In comparison with the baseline, our solution is as not good as the baseline. As the number of watchmen increases, the gap between our solution (best answer) and the baseline tends to stay the same or grow larger. For example, in the 11x11 map, the gaps between them from 2 watchmen to 5 watchmen are $36.7 - 36.1 = 0.6, 4.6, 5.4, 3.4$, the responding proportions of gaps in the baseline are $\frac{0.6}{36.1} = 1\%, 19\%, 30\%, 23\%$. Similarly, in the 13x13 map, the proportions are 18%, 15%, 15%, 20%.

As we hypothesized, the graph heuristics generally performed better than AGS in terms of

Table 5.1: Experiment setting for exploration of proposed efficiency improvements

Number of rounds	Heuristic	ε of pre-pruning	LOS function
30	AGS	1.5	4-way

Table 5.2: Comparison of proposed efficiency improvements

Watchmen	Hash table	Pre-pruning	Exp.	Time	MinMax	Baseline
2	False	False	115.8	325.1	36.7	36.1
	False	True	26.93	2.0	36.7	
	True	False	67.3	4.7	36.7	
	True	True	24.4	0.2	36.7	
3	False	False	468.9	14.5	28.7	24.1
	False	True	291.5	4.05	28.7	
	True	False	297.9	0.8	28.7	
	True	True	193.3	0.3	28.7	
4	False	False	1019.7	14.6	23.5	18.1
	False	True	354.8	2.0	23.5	
	True	False	593.7	0.6	23.5	
	True	True	267.0	0.4	23.5	
5	False	False	1281.0	4.75	17.9	14.5
	False	True	960.7	2.5	17.9	
	True	False	886.5	0.6	17.9	
	True	True	685.8	0.5	17.9	

the number of expansions and the MinMax value. However, the difference between these three functions was not particularly significant. This may be due to the limited size of the maps. Although the graph heuristic functions have a smaller number of expansions, they did not have a clear speed advantage over AGS, which may be due to the cost of building G_{DLS} and the complexity of the algorithms used to calculate the heuristic values $h(u)$.

5.2 Evaluation with Suboptimal WRP-A* solvers

5.2.1 Exploration of Weighted A* search

In this section, we explore the effect of changing the weight of WA* on the time and solution quality. To examine the influence of the number of watchmen on the weight change, we conducted experiments with 3, 4 and 5 watchmen respectively on a 21x21 grid map (shown in Figure 5.4(a)). The experimental settings are given in Table 5.5, and the change in the MinMax criterion with weight and the change in total running time with weight are shown in Figure 5.2 and Figure 5.3 respectively. The horizontal axis represents the weight (1, 4, 6, 8, 10, 12, 14, 16, 18), and the vertical axis represents MinMax value or total running time. Observing the variation in MinMax with weight, The gradient of the line varies greatly at first and then

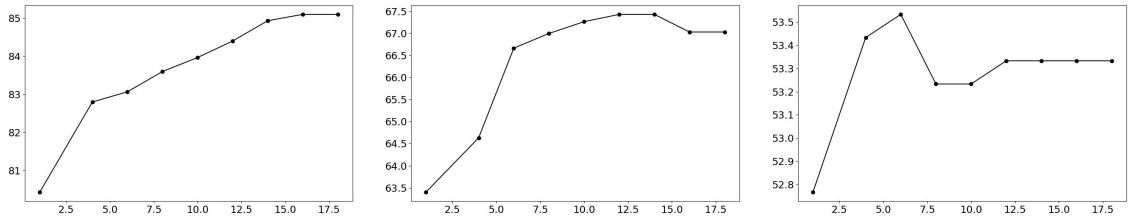
Table 5.3: Experiment setting for comparison with baseline on small maps

Number of rounds	ε of pre-pruning	LOS function
50	1.5	4-way
Pre-pruning	Hash table structure	
True	True	

Table 5.4: Comparison between baseline and optimal WRP-A*

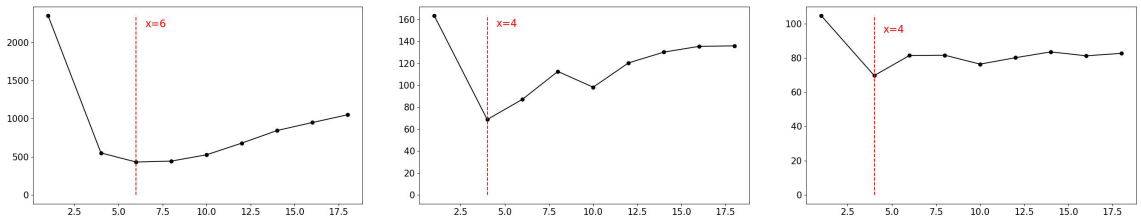
		map 11x11				map 13x13			
Watchmen	Heuristic	Exp.	Time	MinMax	Baseline	Exp.	Time	MinMax	Baseline
2	AGS	**24.4**	**0.1**	**36.7**		155.9	7.1	51.2	
	MST	**24.4**	0.3	**36.7**	36.1	53.8	**5.0**	**50.9**	43.0
	TSP	32.1	0.8	37.3		**53.4**	6.0	**50.9**	
3	AGS	193.3	**0.2**	**28.7**		**152.4**	**1.0**	33.0	
	MST	199.1	0.3	**28.7**	24.1	259.9	1.4	**32.9**	28.6
	TSP	**154.9**	0.8	29.4		259.8	2.0	**32.9**	
4	AGS	267.0	**0.2**	**23.5**		839.9	0.9	25.0	
	MST	265.8	0.3	23.6	18.1	580.5	**0.8**	**24.8**	21.5
	TSP	**259.6**	0.6	24.1		**578.1**	1.0	**24.8**	
5	AGS	685.8	**0.3**	**17.9**		589.7	0.6	21.1	
	MST	725.0	0.4	**17.9**	14.5	521.4	0.6	**20.7**	17.2
	TSP	**589.0**	0.6	18.2		**519.0**	0.8	**20.7**	

tends to become stable. We believe this is because when h is large, starting from a certain boundary point, the expansion of nodes depends entirely on $h(u)$ and is independent of $g(u)$, so the impact on MinMax becomes smaller. This is because, for WRP-A* solver, the node ready to be expanded is the same regardless of how big h is. We can roughly see that there is a generally positive correlation between weight and MinMax. As for the total running time, the weight value for the minimal total running time is slightly different (6, 4, 4) for different numbers of watchmen. The total running time decreases significantly when the weight is below 4. When the weight becomes larger, the lines in the three figures tend to become stable, likely for the same reason as the stability in MinMax mentioned earlier.



(a) 3 watchmen (b) 4 watchmen (c) 5 watchmen

Figure 5.2: MinMax objective value vs. weight



(a) 3 watchmen (b) 4 watchmen (c) 5 watchmen

Figure 5.3: Total running time vs. weight

Table 5.5: Experiment setting of exploration of weighted A* search

Number of rounds	ε of pre-pruning	Heuristic	LOS function
30	1.5	AGS	4-way
Hash table structure	IW	WR	Pre-pruning
True	True	True	True

5.2.2 Comparison with Baseline on Larger Map

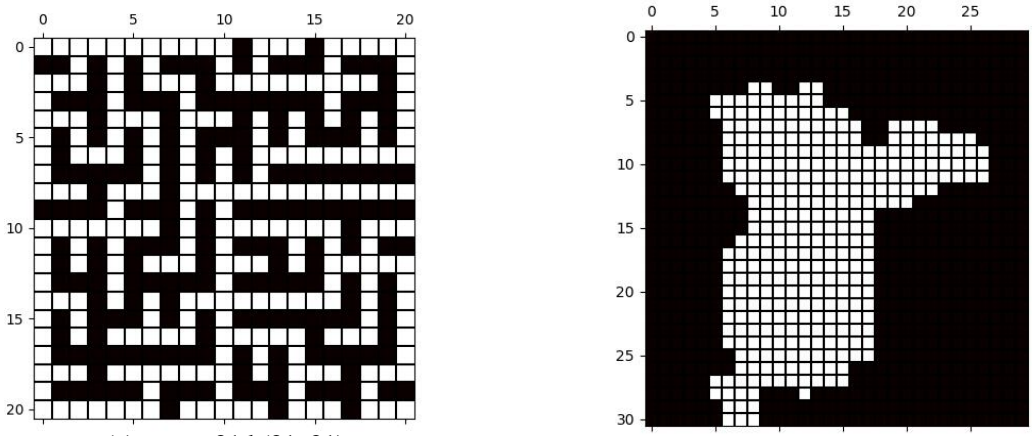
In this section we aim to evaluate our algorithm and compare the solution quality with the baseline. The two maps (lak_101d and maze_21d) we used are from the *moveingai* repository shown in Figure 5.4 [28]. Due to computing resource and time constraints, we only conducted 30 rounds on both maps, using all acceleration measures during the search process. The experimental results are given in Table 5.7. On the maze_21d map, we observed similar results as before, with the graph heuristics performing well in terms of time and number of expansions. The proportion of gaps between our solution and the baseline from 3 to 6 watchmen are 2%, 6%, 9% and 9%, which demonstrates a relatively good performance of the suboptimal solver. However, there is still little difference in the time and solution quality between the different heuristics. The main difference between these two maps is not their size, but rather the presence of open areas. Specifically, the lak_101d map has a large open area that includes all empty cells. As we mentioned in Section 2.3.4, in this situation, AGS is expected to have a significant advantage on this map, which is confirmed by the experimental results.

For the graph heuristics, we were only able to get a rough answer only with 5 watchmen. All other experiments with 3, 4, and 6 watchmen failed due to time constraints. while we were able to solve some instances within the time limit, there were some extreme instances that resulted in a significant delay in obtaining the answer, leading to a tremendous increase in the average time. e.g., in the case of 5 watchmen, the average solving time was 932.6, but the minimum solving time was 46.9 and the maximum solving time was as high as 6939.2. This extreme instance had a significant impact on the average computation time and was also the main reason that the experiments with different numbers of watchmen timed out. Thus, our algorithm has robustness issues on large maps.

Additionally, we found that when a map has a large open area, the node expansion of our algorithm significantly increases and the number of nodes generated increases with the number of watchmen. As for the reason why the solution quality may not be sufficient, we suspect that in such an open area, there will inevitably be a larger number of LOS duplication under the 4-way LOS setting, and the degree of duplication becomes more severe as the number of watchmen increases. Two samples are shown in Figure 5.5. It is advisable to choose an appropriate number of watchmen to reduce LOS duplication.

Table 5.6: Experiment setting for comparison with baseline on large grid-map

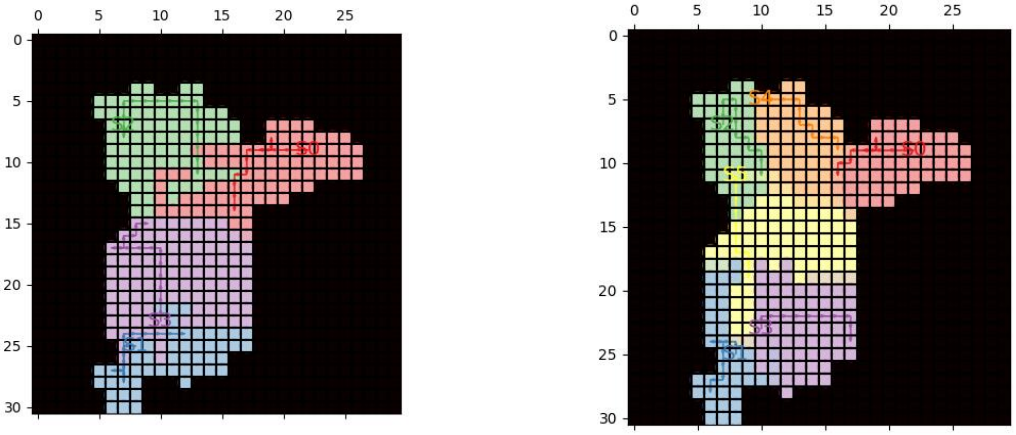
Number of rounds	ε of pre-pruning	w of WA*	LOS function
30	1.5	10	4-way
Hash table structure	IW	WR	Pre-pruning
True	True	True	True



(a) maze_21d (21x21)

(b) lak_101d (30x31)

Figure 5.4: Larger maps for experiment



(a) 4 watchmen solution

(b) 6 watchmen solution

Figure 5.5: A sample of solutions on large map

Table 5.7: Comparison between baseline and suboptimal WRP-A*

		maze_21d (21x21)				lak_101d (30x31)			
Watchmen	Heuristic	Exp.	Time	MinMax	Baseline	Exp.	Time	MinMax	Baseline
3	AGS	740.0	18.4	83.4	80.8	**11799.0**	**77.2**	**24.4**	18.09
	MST	487.6	**3.0**	85.0		—	>4200	—	
	TSP	**439.7**	3.4	**82.7**		—	>4200	—	
4	AGS	551.1	3.5	65.2	60.6	**16744.9**	**108.9**	**19.7**	13.6
	MST	560.5	2.0	67.2		—	>4200	—	
	TSP	**422.4**	**1.5**	**64.3**		—	>4200	—	
5	AGS	1006.4	2.6	**52.7**	48.5	20965.4	**63.7**	18.1	10.9
	MST	**486.6**	**1.6**	53.6		**20028.4**	589.1	17.2	
	TSP	960.8	2.6	53.6		20616.2	932.6	**16.9**	
6	AGS	1866.4	3.6	**44.0**	40.4	**24328.0**	**62.3**	**16.8**	9.0
	MST	**879.7**	**2.3**	45.1		—	>4200	—	
	TSP	1942.7	4.4	45.6		—	>4200	—	

Chapter 6

Conclusion and Future Work

In this thesis, we solved the MWRP with a new method consisting of a multi-clustering algorithm and a generate-and-repair search algorithm. Our method achieved a solution quality that is nearly as good as the baseline, demonstrating its feasibility and potential for further development. We also demonstrated the effectiveness of our proposed efficiency measures in saving time. By comparing the results of different heuristics on different types of maps, we found that on a map with a large open area, the algorithm with AGS heuristic is significantly faster than other heuristics in MWRP. In addition, we explored the effect of changing the weight of the WA* algorithm on time and performance, providing useful guidance for selecting an appropriate weight depending on the situation.

We felt that there is still much work to be done to improve our method and potential future directions include:

First, designing a new evaluation function for the MinSum criterion, the current evaluation function is more like a universal one. We want the evaluation function specifically tailored for the MinSum or MinMax criteria.

Second, improving the conditions for expanding new nodes with the iterative repairing algorithm, particularly in the case of maps with large open areas where many redundant nodes may be expanded, consuming time and computing resources. For example, a smarter algorithm could be developed to more selectively choose which nodes deserve expansion, similar to the pre-pruning measure we proposed.

Finally, ensuring the robustness of the algorithm to prevent it from getting bogged down in overly complex calculations, which can cause overrun. This could be achieved by estimating the calculation time of the WRP-A* algorithm or by avoiding assigning too many cells to a single watchman at once.

Acknowledgement

I would like to express my deep thank to my supervisor, Professor Fukunaga Alex, for his insightful guidance and great help throughout the writing of this thesis. He consistently supported my research and I benefited a lot from him through weekly meetings. Without his help, I would have a hard time completing my research. I would also thank my parents for their consideration and financial help under the tough period.

Bibliography

[1] David Arthur and Sergei Vassilvitskii. k-means++: The advantages of careful seeding. Technical report, Stanford, 2006.

[2] Wei-Pang Chin and Simeon Ntafos. Optimum watchman routes. In *Proceedings of the second annual symposium on Computational geometry*, pages 24–33, 1986.

[3] Tim Danner and Lydia E Kavraki. Randomized planning for short inspection paths. In *Proceedings 2000 ICRA. Millennium Conference. IEEE International Conference on Robotics and Automation. Symposia Proceedings (Cat. No. 00CH37065)*, volume 2, pages 971–976. IEEE, 2000.

[4] Marco Dorigo and Luca Maria Gambardella. Ant colonies for the travelling salesman problem. *biosystems*, 43(2):73–81, 1997.

[5] Jan Faigl. Approximate solution of the multiple watchman routes problem with restricted visibility range. *IEEE transactions on neural networks*, 21(10):1668–1679, 2010.

[6] Jan Faigl, G Klacnar, Drago Matko, and Miroslav Kulich. Path planning for multi-robot inspection task considering acceleration limits. In *Proceedings of the fourteenth International Electrotechnical and Computer Science Conference ERK 2005*, pages 138–141, 2006.

[7] Robert W Floyd. Algorithm 97: shortest path. *Communications of the ACM*, 5(6):345, 1962.

[8] Robert Geisberger, Peter Sanders, Dominik Schultes, and Daniel Delling. Contraction hierarchies: Faster and simpler hierarchical routing in road networks. In *International workshop on experimental and efficient algorithms*, pages 319–333. Springer, 2008.

[9] Michel Gendreau, Gilbert Laporte, and Frédéric Semet. A tabu search heuristic for the undirected selective travelling salesman problem. *European Journal of Operational Research*, 106(2-3):539–545, 1998.

[10] Héctor González-Banos. A randomized art-gallery algorithm for sensor placement. In *Proceedings of the seventeenth annual symposium on Computational geometry*, pages 232–240, 2001.

[11] Peter E Hart, Nils J Nilsson, and Bertram Raphael. A formal basis for the heuristic determination of minimum cost paths. *IEEE transactions on Systems Science and Cybernetics*, 4(2):100–107, 1968.

[12] Robert C Holte, Ariel Felner, Jack Newton, Ram Meshulam, and David Furcy. Maximizing over multiple pattern databases speeds up heuristic search. *Artificial Intelligence*, 170(16-17):1123–1136, 2006.

[13] Giorgos D Kazazakis and Antonis A Argyros. Fast positioning of limited-visibility guards for the inspection of 2d workspaces. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, volume 3, pages 2843–2848. IEEE, 2002.

[14] Teuvo Kohonen. The self-organizing map. *Proceedings of the IEEE*, 78(9):1464–1480, 1990.

[15] Joseph B Kruskal. On the shortest spanning subtree of a graph and the traveling salesman problem. *Proceedings of the American Mathematical society*, 7(1):48–50, 1956.

[16] Miroslav Kulich, Jan Faigl, and Libor Preucil. Cooperative planning for heterogeneous teams in rescue operations. In *IEEE International Safety, Security and Rescue Rototics, Workshop, 2005.*, pages 230–235. IEEE, 2005.

[17] Fajie Li and Reinhard Klette. An approximate algorithm for solving the watchman route problem. In *International Workshop on Robot Vision*, pages 189–206. Springer, 2008.

[18] Yaakov Livne, Dor Atzmon, Shawn Skyler, Eli Boyarski, Amir Shapiro, and Ariel Felner. Optimally solving the multiple watchman route problem with heuristic search. In *Proceedings of the International Symposium on Combinatorial Search*, volume 15, pages 302–304, 2022.

[19] J MacQueen. Classification and analysis of multivariate observations. In *5th Berkeley Symp. Math. Statist. Probability*, pages 281–297, 1967.

[20] Eli Packer. Computing multiple watchman routes. In *International Workshop on Experimental and Efficient Algorithms*, pages 114–128. Springer, 2008.

[21] Eli Packer. Robust geometric computing and optimal visibility coverage, 2008.

[22] Ira Pohl. Heuristic search viewed as path finding in a graph. *Artificial intelligence*, 1(3-4):193–204, 1970.

[23] Robert Clay Prim. Shortest connection networks and some generalizations. *The Bell System Technical Journal*, 36(6):1389–1401, 1957.

[24] Raimund Seidel. A simple and fast incremental randomized algorithm for computing trapezoidal decompositions and for triangulating polygons. *Computational Geometry*, 1(1):51–64, 1991.

[25] Shawn Seiref, Tamir Jaffey, Margarita Lopatin, and Ariel Felner. Solving the watchman route problem on a grid with heuristic search. In *Proceedings of the international conference on automated planning and scheduling*, volume 30, pages 249–257, 2020.

[26] Shawn Skyler, Dor Atzmon, Tamir Yaffe, and Ariel Felner. Solving the watchman route problem with heuristic search. *Journal of Artificial Intelligence Research*, 75:747–793, 2022.

[27] Samerkae Somhom, Abdolhamid Modares, and Takao Enkawa. Competition-based neural network for the multiple travelling salesmen problem with minmax objective. *Computers & Operations Research*, 26(4):395–407, 1999.

[28] Nathan R Sturtevant. Benchmarks for grid-based pathfinding. *IEEE Transactions on Computational Intelligence and AI in Games*, 4(2):144–148, 2012.

[29] Csaba D Toth, Joseph O'Rourke, and Jacob E Goodman. *Handbook of discrete and computational geometry*. CRC press, 2017.