Optimally Solving the Multiple Watchman Route Problem with Heuristic Search

Yaakov Livne
Ben-Gurion University, Israel
livnya@post.bgu.ac.il

Dor Atzmon
Ben-Gurion University, Israel
Royal Holloway University of London, UK
dorat@post.bgu.ac.il,dor.atzmon@rhul.ac.uk

Shawn Skyler
Ben-Gurion University, Israel
shawn@post.bgu.ac.il

Eli Boyarski
Ben-Gurion University, Israel
boyarske@post.bgu.ac.il

Amir Shapiro
Ben-Gurion University, Israel
ashapiro@bgu.ac.il

Ariel Felner
Ben-Gurion University, Israel
felner@bgu.ac.il

ABSTRACT

In the *Watchman Route Problem* (WRP), the task is to find a path for a watchman agent such that all locations in the given map will be visually seen by the watchman at least once during the path traversal. Recently, the problem has been optimally solved on a grid map using heuristic search. In this paper, we extend this work to the case of multiple agents. We call this problem the *Multiple Watchman Route Problem* (MWRP). In MWRP, the task is to find a path for each watchman such that each location on the map will be seen by at least one watchman. We optimally solve MWRP with heuristic search for two different objective functions with a number of A*-based variants, including an enhanced branching mechanism. We then provide an experimental study on these methods and on other attributes of this problem.

KEYWORDS

Multi-Agent, Heuristic Search, Watchman Route Problem

ACM Reference Format:
Yaakov Livne, Dor Atzmon, Shawn Skyler, Eli Boyarski, Amir Shapiro, and Ariel Felner. 2023. Optimally Solving the Multiple Watchman Route Problem with Heuristic Search. In *Proc. of the 22nd International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2023), London, United Kingdom, May 29 – June 2, 2023*, IFAAMAS, 9 pages.

1 INTRODUCTION AND OVERVIEW

In the *Watchman Route Problem* (WRP), the task is to find a path for a traveling agent (called the *watchman*) on a map such that any location on the map is visually seen from at least one location on the path [5, 12]. As a result, a watchman that follows that path will be able to see the entire set of locations of the map. In WRP, the sight of the agent is modeled using a *Line-of-Sight* (LOS) function, which determines whether any given two cells can visually see each other. Trivial LOS functions on grids are the cardinal lines (4-way), possibly combined with diagonal lines (8-way). A non-trivial LOS function for general graphs is a *transmission function* that indicates for each vertex which are the vertices that can receive the transmission. WRP is applicable in many real-life scenarios, such as planning a path for visiting a museum and seeing all items, finding an efficient path for watering the garden, etc.

Optimally solving WRP was proven NP-hard for polygons [5]. Recently, WRP was solved optimally on grid maps using heuristic search while proving that it is also NP-hard [12, 13]. Hereafter, we refer to that paper as S20. S20 developed a variant of A* called WRP-A*, which is executed on a state-space derived from the problem. A simple admissible heuristic function, called $h_{Singleton}$, was proposed. It uses the cost of reaching a location that sees the farthest cell. More complex admissible heuristics were based on abstraction of the grid to the *Disjoint Line-of-Sight Graph* (G_{DLS}). The best heuristic was based on a solution to a variant of the *Traveling Salesman Problem* (TSP) applied on G_{DLS}. Yaffe et al. (2021) extended this work and developed suboptimal variants of WRP-A*, which run faster than the optimal solver with a relatively minor increase in solution cost.

In this paper, we extend WRP to the case of multiple agents (watchmen) and denote this problem as *Multiple Watchman Route Problem* (MWRP). In MWRP, the task is to find a path for each of the agents from its start cell through the grid such that all empty cells in the map will be covered by LOS from at least one cell of one of the paths. While in WRP the shortest path is desired, in MWRP, we consider two objective functions: (1) The sum of the costs (SOC) of all the paths of the agents, and (2) *Makespan* (MKSP) – the cost of the longest path among these paths.

MWRP is applicable in WRP applications where multiple agents exist. For example, consider a museum with guards that need to tour the venue and check that all the exhibits are in good shape before opening the museum to the public. These guards want to arrive to work as late as possible and still complete their tour on time. The paths returned from solving MWRP can be calculated once and be used by these guards for many years.[1] Note that while MWRP has multiple agents, it is a classical combinatorial search problem with a fully structured state-space which is solved offline. It is not to be confused with online multi-agent systems where agents have their own computing power, sensing and communication paradigms, and error possibilities.

We formalize MWRP as a search problem and develop an A*-like algorithm (MWRP-A*) for optimally solving MWRP. We then propose two admissible heuristic functions for MWRP-A*, which generalize two of the heuristics that were suggested by S20 for WRP, namely, $h_{Singleton}$ and h_{TSP}. We then develop a method that intelligently combines these two heuristics using Lazy A* [17]. Additionally, we propose a unique method for enhancing the search,

Proc. of the 22nd International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2023), A. Ricci, W. Yeoh, N. Agmon, B. An (eds.), May 29 – June 2, 2023, London, United Kingdom.

[1] In fact, our work was sponsored by an industrial company that has very similar tasks (not in a museum though).

called *Expanding Border* (EB). When EB expands a state s in the search, it does not branch to the neighbors of s but directly jumps to farther cells that reveal new unseen cells. This significantly reduces the size of the search tree without losing any possible solution. We experimentally show the pros and cons of our different methods. In addition, we study the effect of different policies for selecting the start cells, the effect of increasing the number of agents on the behaviors of the algorithms and on the optimal cost, and the effect of requiring the agents to return to their start cells.

Our work is applicable to any general graph and is robust across any LOS function. To be focused, we follow S20 and demonstrate the problem on grid maps and experiment with the common *BresLos* LOS function (see below).

2 RELATED WORK

There are many important problems that are related to WRP. WRP itself was researched in the field of computational geometry [5] (mainly on polygons with different characteristics). Other related problems are the *Simultaneous Localization And Mapping* problem (SLAM) [1, 16], *inspection planning* [7], and the *Art Gallery problem* (AGP) [8]. We refer the reader to S20 for a broad survey on these problems and their relation to WRP.

The *Multiple Traveling Salesman Problem* (MTSP) extends TSP for multiple agents [9]. MTSP is identical to MWRP with a trivial LOS function that returns the input cell only, i.e., an agent needs to arrive at a location to see it. One approach for solving MWRP [4, 6, 10] first selects a set of locations on the graph that covers the entire graph by their LOS. Then, an MTSP solver is applied on this set of locations. Somhom et al. (1997) solved MWRP by first abstracting the map using *Self-Organizing Maps* (SOM). Then, they found paths that cover all the map by building a neural-network on the abstract map. In contrast to our work, all these approaches find circular paths and do not guarantee to return an optimal solution. While we do not require circular paths, we show how our methods can be easily adjusted to find optimal circular solutions.

MWRP is an offline multi-agent coordination problem, and thus can also be seen as a member of the Multi-Agent Path Finding problems [15], where collision-free paths should be computed. In MWRP, we do not consider collisions, namely, we assume that each location is large enough to contain multiple agents. Also, collisions only affect the solutions if the domain is condensed with many agents. This is not the case in MWRP where usually the environment is large and only a few agents exist. Finally, our algorithms below may be modified to work when collisions are not allowed.

3 PROBLEM DEFINITION

The input for MWRP is a grid map. The set of empty (traversable) cells is denoted by M. Blocked cells are denoted as *obstacles*. We are also given a set of agents $A = \{a_1, \ldots, a_m\}$, where each agent a_i has a start cell $start_i \in M$. Time is discrete and between two consecutive timesteps each agent located in cell s can perform a *move action* to one of its empty neighboring cells (denoted as $N(s)$), along the four cardinal directions (up, down, left, right).

All agents have a *Line-of-Sight* function (LOS) which determines whether any given two cells can visually see each other and it can be any arbitrary function. S20 experimented with three possible LOS

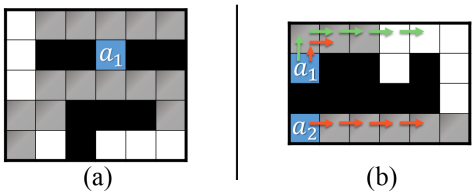

Figure 1: (a) *BresLos* (b) Optimal MKSP (red), SOC (green)

functions on grids: 4-way, 8-way, and *Bresenham LOS* (*BresLos*) [3]. *BresLos* is commonly used in computer graphics, video games, and bitmap pictures. It is perhaps the most suitable LOS function that discretizes real-world continuous domains into grids and simulates a continuous field of view. Intuitively, it approximates a straight line between two cells that does not pass through any obstacles. Thus, it allows to see more cells beyond the cardinal and diagonal lines. The exact definition of *BresLos* is complex and is given by Bresenham (1965). The gray cells in Figure 1(a) have LOS to a_1 according to *BresLos*, where black cells are obstacles. S20 showed that the relative performance of different algorithms is robust across these LOS functions. So, following Yaffe et al. (2021), in this paper, we only work with *BresLos*. We also assume a symmetric LOS, i.e., $s_1 \in LOS(s_2) \Leftrightarrow s_2 \in LOS(s_1)$. Our work can be easily adjusted to other (possibly asymmetric) LOS functions or to cases where agents can perform other movements, e.g. diagonal movements or more complex moves [11].

The output for MWRP is a set of paths $\pi = \{\pi_1, \ldots, \pi_m\}$, where path $\pi_i = \langle s_0 = start_i, ..., s_k \rangle$ for agent $a_i \in A$ is a sequence of neighbouring cells starting from $start_i$. The requirement for π is that for every cell $s \in M$, there is LOS from at least one cell from at least one path. That is, if all agents follow their paths then every cell will be seen by at least one of the agents.

The cost $c(\pi_i)$ of a single path π_i is the number of actions performed in π_i ($c(\pi_i) = |\pi_i| - 1$). A set of paths π is *optimal* iff it has the minimal cost $c(\pi)$ among all sets of paths, according to a given objective function for aggregating the costs of multiple paths. We consider the following objective functions, commonly used in multi-agent problems [15]: **(1) Sum of Costs** (SOC), which is the sum of the cost of all paths in π, i.e., $SOC(\pi) = \sum_{\pi_i \in \pi} c(\pi_i)$. **(2) Makespan** (MKSP), which is the cost of the longest path in π, i.e., $MKSP(\pi) = \max_{\pi_i \in \pi} c(\pi_i)$. We denote the shortest distance between two cells $s_1, s_2 \in M$ by $d(s_1, s_2)$.

Following S20, we assume that agents do not return to their start cells and the whereabouts of agents after all cells have been seen is of no importance (they can leave via the closest door, they can stand still waiting for new tasks, or they can destroy themselves). S20 showed that the problem can be simply reduced to handle the case where agents need to return to their start locations. Indeed, we show below how our methods can find such circular paths.

Figure 1(b) shows an example of an instance with two agents. Blue cells represent the start cells of the agents, grey cells are those that have LOS to/from the start cells (with *BresLos*), and white cells represent unseen cells. The red arrows (π_r) and the green arrows (π_g) demonstrate two possible solutions for this example. Note that, in π_r the middle cells in the rightmost column are seen by a_2 from

the bottom row. By contrast, in π_g, agent a_2 stands still in its start cell and these cells are seen by a_1 from the top row. Observe that $SOC(\pi_r) = 6$ and $SOC(\pi_g) = 5$. Thus, π_g is better (and optimal) for minimizing SOC. However, $MKSP(\pi_r) = 4$ and $MKSP(\pi_g) = 5$, and hence π_r is better (and optimal) for minimizing MKSP.

4 MWRP AS A SEARCH PROBLEM

To solve MWRP with heuristic search, we first define the corresponding search tree which generalizes that of S20.

Node. A *node* is the pair $\langle L, seen\rangle$. $L = \{l_1, \ldots, l_m\}$ is a set of locations, where $l_i \in M$ represents the current location of agent a_i. $seen \subseteq M$ is the set of cells that have already been seen by at least one of the agents during its movement. The complement of *seen* is the *unseen* set (the cells that have not been seen yet by any of the agents).

Root Node. The root node (*Root*) contains the start cells of all agents ($Root.L = \{start_1, \ldots, start_m\}$) and $Root.seen$ contains all cells that are seen from these start cells ($Root.seen = \bigcup_{start_i} LOS(start_i)$).

Goal Node. A node n is a goal node iff $n.seen = M$. Note that the agents can finish the task in many sets of locations ($n.L$), as long as all cells have been seen.

Successor Function. Recall that each agent a_i positioned in l_i can perform a move action to an empty neighboring cell in $N(l_i)$. Additionally, as the solution may contain paths of different lengths, we add a *terminate action* which costs 0 and prevents the agent from further exploration (i.e., it is out of the game). For each node n, we divide A and L into two groups: A^+ and L^+ refer to non-terminated agents and A^- and L^- refer to terminated agents. Let $n = \langle L, seen\rangle$ be the node selected for expansion and let N_i represent the set of locations that agent a_i can reach by performing an action from its current location $l_i \in n.L$. If agent a_i has not terminated yet ($a_i \in A^+$), the agent can either perform a move action to one of its neighbors cells $N(l_i)$ or perform a terminate action, i.e., $N_i = N(l_i) \cup \{t_i\}$, where t_i indicates that a_i has just terminated (it can be seen as a virtual cell where the agent can no longer move). If agent a_i has already terminated ($a_i \in A^-$), then $N_i = \{t_i\}$. When n is expanded, a new node n' is generated for each possible combination in the Cartesian product of actions. However, we exclude the combination of all agents performing a terminate action because it will lead to a dead-end. Formally, the set of successors of n is $\{\{N_1 \times \cdots \times N_m\} \setminus \{t_1, t_2, \ldots, t_m\}\}$ (coupled with the corresponding *seen* sets that are determined according to the new seen cells).

For example, for two non-terminated agents, where each can move to its four neighbors or perform a terminate action, there are $5 \times 5 - 1 = 24$ possible combinations of actions. For each such successor, locations of non-terminated agents are placed in L^+ and the last locations of terminated agents are placed in L^-. The pseudo-code for the expansion procedure is provided in Algorithm 1.

5 MWRP-A*: A*-LIKE SEARCH ALGORITHM

Following S20 (which introduced WRP-A*), we propose MWRP-A*, an A*-like search algorithm for optimally solving MWRP. In MWRP-A*, each node n is associated with $g(n)$ and $h(n)$. $g(n)$ represents the cost of reaching n from the root, and is calculated according to SOC or MKSP of $n.L$ (naturally, $g(Root) = 0$). $h(n)$ estimates the remaining cost for reaching a goal from n and we introduce below

Algorithm 1: Basic expansion

```
1 BE(Node n)
2     foreach l_i ∈ n.L do
3         Init N_i
4         if l_i ∈ n.L^+ then
5             N_i ← N(l_i)
6         N_i ← N_i ∪ {t_i}
7     return {{N_1 × ··· × N_m} \ {t_1, t_2, ..., t_m}}
```

a number of admissible heuristic functions. As usual, we use *Open* and *Closed* lists for tracking the search process. In MWRP-A*, *Open* is ordered according to $f(n) = g(n)+h(n)$ of each node n. MWRP-A* extracts from *Open* the node n with the lowest f-value and performs a goal test. If n is a goal, the search halts and a solution is returned. Otherwise, the search activates the successor function on node n and for each new generated node n' it sets $f(n') = g(n') + h(n')$. If n' is a duplicate or dominated node (explained next), it is discarded. Otherwise, n' is inserted to *Open*. Finally, n is moved to *Closed* and another expansion cycle is performed.

Duplicate/dominance Detection. Note that the paths associated with a given node n are not directly described in the node, and can be derived from the current branch of n in the search tree. This enables us to prune duplicates, which have the same set of locations and the same seen set as they represent the same situation for the search task. We extend this to a *dominance relation* as follows. Let n_1 and n_2 be two nodes (in our case one of them is a newly generated node and the other is a node in $Open \cup Closed$). If (1) $n_1.L^+ \subseteq n_2.L^+$, (2) $n_1.seen \subseteq n_2.seen$, and (3) $g(n_2) \leq g(n_1)$ then n_1 can be discarded. Note that, $n_1.L^+ \subseteq n_2.L^+$ means any permutation of the individual agents into the same set of locations. For example, two agents a_1 and a_2 that start in locations $start_1$ and $start_2$ may be in locations s_1 and s_2 in $n_1.L^+$, but in locations s_2 and s_1 in $n_2.L^+$ and still $n_1.L^+ = n_2.L^+$. This is done before h is computed.

6 HEURISTIC FUNCTIONS FOR MWRP

In this section, we define admissible heuristic functions that estimate the remaining cost for a given node n.

6.1 Singleton Heuristic

Given a cell s and an agent a_i (located in cell l_i), the minimum cost for a_i of moving and seeing cell s is:

$$q(s, l_i) = \min_{w \in LOS(s)} d(l_i, w) \tag{1}$$

As each of the agents can be the one selected for seeing cell s, the lowest cost for that purpose is:

$$q(s) = \min_{l_i \in n.L^+} q(s, l_i) \tag{2}$$

In MWRP all cells need to be seen. Thus, we calculate the cost of seeing each unseen cell (in $n.unseen$) using Equation 2 and take the maximum value as a heuristic:

$$h_{Singleton}(n) = \max_{s \in n.unseen} q(s) \tag{3}$$

As each cell in $n.unseen$ needs to be seen, it is easy to see that $h_{Singleton}$ is admissible for both SOC and MKSP. We note that

$h_{Singleton}$ proposed by S20 is a special case where only one agent exists. Thus, it omits Equation 2.

6.2 MTSP Heuristic

S20 abstracted the grid into a graph and showed that a TSP solution on that graph is an admissible heuristic to WRP. We next generalize this heuristic and use the solution to MTSP [9] (where multiple traveling agents exist) as an admissible heuristic to MWRP.

Graph Abstraction for the MTSP Heuristic. We say that two cells in M are *LOS-disjoint* if there is no cell that they both see. We say that a set of cells P is *LOS-disjoint* if every pair of cells in P is *LOS-disjoint*. For node n, a *Disjoint LOS Graph* $G_{DLS}(n) = (V, E)$ is a weighted, directed graph which is abstracted from the grid. G_{DLS} is built by first identifying a set of *LOS-disjoint* cells $P \subseteq n.unseen$. Each cell in P is called a *pivot*. There are different methods for choosing a set of pivots. We suggest one such method below. For each pivot $p \in P$, we refer to all cells in $LOS(p)$ as *watchers* of p.

Figure 2(a) depicts a problem instance with two agents a_1 and a_2, which start in the blue cells ($L = \{a_1, a_2\}$). Let node $n = \langle L, seen \rangle$ (n is the root in our case) be the node for which we calculate the MTSP heuristic. The grey cells can be seen from these start locations and are in $n.seen$. Figure 2(b) shows an example of selected pivots p_1, p_2, p_3 (green cells) and their watchers (yellow cells).

In G_{DLS}, we create a vertex for each pivot (pivot vertex). We add an undirected edge between every two pivots p_i and p_j with a weight of $d(w_i, w_j)$ where w_i and w_j are watchers of p_i and p_j with the minimal distance between them among all pairs of watchers of p_i and p_j. Figure 2(c) presents the constructed G_{DLS} (the pivot vertices are colored green). As the shortest path between a watcher of p_1 and a watcher of p_2 is 4, the cost of the edge between those vertices in G_{DLS} is 4. The current locations of the agents $n.L$ are also added as vertices to G_{DLS} (agent vertices, blue). We add a directed edge from each agent vertex (located in l) to each pivot vertex with a cost of $d(l, w)$ where w is a watcher of pivot p with the shortest distance to l. In our example, we add a directed edge from a_1 to p_1 with a weight of 2 (the shortest path from the location of a_1 to a watcher of pivot p_1). Finally, we add a *base vertex* for all agents (V^*, purple) and undirected edges with cost 0 between V^* and any agent vertex. Additionally, we add directed edges with cost 0 from each pivot vertex to V^*. In MTSP, the agents do not have to start from the same vertex. We add V^*, as well as the edges related to V^*, mainly for the EB enhancement described below.

h_{MTSP} uses the optimal solution to the MTSP problem on $G_{DLS}(n)$ as an admissible heuristic. An optimal MTSP solution for our example is shown in Figure 2(d) where both agents start at V^*: agent a_1 goes to see pivot p_1 and agent a_2 goes to see pivot p_3 and then pivot p_2 (both return to V^*).

Theorem 1 (Admissibility). *The cost of an optimal MTSP solution for $G_{DLS}(n)$ is admissible for n in MWRP.*

Proof outline. All pivots are *LOS-disjoint* and they must all be seen. As in S20, visiting a pivot vertex (in G_{DLS}) corresponds to the lowest cost of moving and seeing this pivot in MWRP. The weight of an edge between two vertices in G_{DLS} (excluding V^*) is the shortest distance between some watcher of a pivot and an agent location (or a watcher of another pivot). So, an optimal MTSP solution on $G_{DLS}(n)$ is a lower bound of a solution for MWRP. It might underestimate the real solution because only the pivots are covered by the MTSP. Also, in $G_{DLS}(n)$, traveling between watchers of the same pivot costs 0. □

Solving MTSP. We formulated MTSP as a *constraint programming* problem, as done by Kivelevitch et al. (2013). We force all agents to leave V^* to their start cells. We then used the *CPLEX optimizer* (https://www.ibm.com/analytics/cplex-optimizer) for solving it. However, any MTSP solver can be used here.

Choosing Pivots. In this work, for each node n we chose the pivots with the highest *farness centrality* (FC) [2] as follows: (1) We add all cells in $n.unseen$ to a set S. (2) We calculate $FC(s)$ for each cell $s \in S$, where $FC(s) = \sum_{s' \in S} d(s, s')$. (3) We add the cell s with the highest farness centrality in S to the set of pivots P. (4) We remove from S each cell that is not *LOS-disjoint* with at least one cell in P. (5) We go back to step (3) for choosing another pivot. In our experiments, we limit the maximum number of pivots to six. We tried other ways of choosing pivots and found this method to be the most effective. This is reasonable as it chooses pivots that are far and have a high cost to reach.

6.3 Combining Heuristics

Our two heuristic functions can be considered as a simple heuristic ($h_{Singleton}$) and a complex heuristic (h_{MTSP}). As both are admissible, they can be admissibly combined. Thus, we also consider two known methods. **(1) *Max***, which calculates both heuristic functions for each node and takes their maximum as a heuristic. **(2) *Lazy A**** [17], which first calculates $h_{Singleton}(n)$ for each node and inserts n to *Open*. Then, when n is extracted from *Open*, it calculates $h_{MTSP}(n)$ and inserts n back to *Open* with its new f-value. A node is expanded only if it is extracted from *Open* and both heuristics have already been calculated. The benefit of Lazy A* over the standard Max is that, for many nodes with $f \geq C^*$ (C^* is the cost of the optimal solution), Lazy A* only calculates the simple heuristic, while Max must calculate both heuristics for each generated node.

7 EXPANDING BORDER ENHANCEMENT

In the standard expansion, when node n is expanded, new nodes are generated for each combination of move/terminate actions for the agents. For some of these actions, *seen* may not change as no new information is revealed. We therefore suggest a more advanced expansion mechanism, called the *Expanding Border* mechanism (EB). A pseudo-code for EB is given in Algorithm 2.

In EB, we want to move each agent a_i to a closest location where it can see new cells (that are not in *seen*). Such locations may be farther than the immediate neighbors of l_i. As a result, in EB, in a generated node, different agents may have paths of different costs. Hence, in EB, for each node n, we maintain (in $n.L$) a pair $\langle l_i, c_i \rangle$ of a location l_i and cost c_i — the aggregated cost of the agent from $start_i$ to l_i. We build the set N_{EB_i} that includes all location-cost pairs for agent a_i that satisfy this as follows. For each non-terminated agent $a_i \in A^+$, a breadth-first search (BFS) is executed from location l_i. When the BFS encounters a cell s whose $LOS(s)$ includes some cell that is not in $n.seen$, we add $\langle s, c_i + d(l_i, s) \rangle$ to N_{EBi} and the BFS does not expand s. When the BFS halts, N_{EBi}

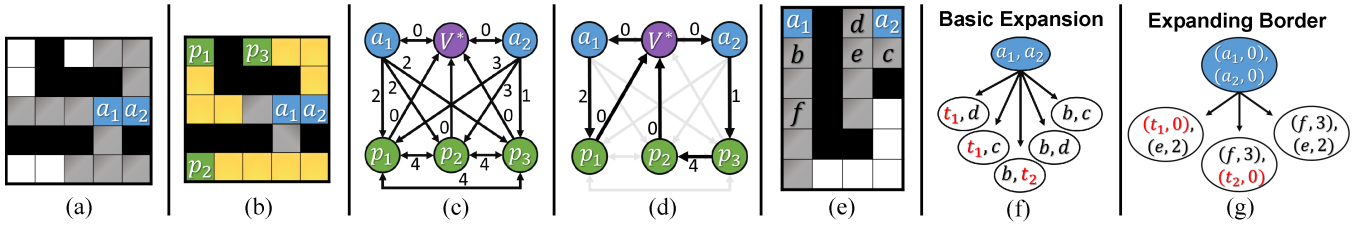

Figure 2: (a-d) MTSP heuristic example. (e-g) Basic Expansion vs. Expanding Border.

Algorithm 2: The EB enhancement

```
1  EB(Node n)
2      foreach ⟨l_i, c_i⟩ ∈ n.L do
3          Init N_EB_i, Open, Closed
4          if ⟨l_i, c_i⟩ ∈ n.L+ then
5              Insert ⟨l_i, 0⟩ into Open
6              while Open is not empty do
7                  ⟨l, c⟩ ← Extract from Open // lowest c
8                  if ∃l' ∈ LOS(l) s.t. l' ∉ n.seen then
9                      N_EB_i ← N_EB_i ∪ ⟨l, c + c_i⟩
10                 else
11                     foreach l' ∈ N(l) do
12                         if l' ∉ Closed then
13                             Insert ⟨l', c + 1⟩ into Open
14                 Insert l into Closed
15         N_EB_i ← N_EB_i ∪ {t_i}
16     return {{N_1 × ··· × N_m}\{t_1, t_2, ..., t_m}}
```

includes a perimeter of cells (border) around l_i such that they all see new cells that are not currently in *seen*. For non-terminated agents, we also add a terminate action to N_{EBi}. For terminated agents, we just set $N_{EBi} = \{t_i\}$. Then, a new node n' is generated for each possible combination of the Cartesian product of pairs in $\{\{N_{EB1} \times \cdots \times N_{EBm}\} \backslash \{t_1, t_2, \ldots, t_m\}\}$ (again, we do not allow to terminate all agents).[2]

For example, Figure 2(e) presents a problem instance with two agents located in start cells a_1 and a_2 (denoting agents and cells). The neighbors of cells a_1 and a_2 are $\{b\}$ and $\{c, d\}$, respectively. Figure 2(f,g) shows the basic expansion method and EB for expanding the root node. Red fonts denote a terminated agent. In EB, a BFS for each agent is performed to see the white cells, which are not in *seen*. For a_1, the BFS halts with $(f, 3)$ and, for a_2, the BFS halts with $(e, 2)$. Thus, the root is expanded accordingly. Three nodes are generated including those where one of the agents is terminated.

Singleton Heuristic. Here, we differentiate the treatment of SOC from that of MKSP.

For SOC, the g-costs are $g(n) = \sum_{\langle l_i, c_i \rangle \in n.L} c_i$. Recall that we defined (Equation 1) the cost $q(s, l_i) = \min_{w \in LOS(s)} d(l_i, w)$ of moving and seeing a given cell s from location l_i. As each of the agents can be the one selected for seeing cell s, the lowest cost for that purpose is:

$$q_{EB}(s) = \min_{\langle l_i, c_i \rangle \in n.L^+} q(s, l_i)$$

[2] `S20` used a different mechanism (JF). That mechanism removes some pivots if they are on the way to other pivots. This removal is only possible for a single agent.

In MWRP, all cells need to be seen. Thus, we calculate the cost of seeing each unseen cell using $q_{EB}(s)$ and take the maximum value as a heuristic:

$$h_{SingletonEB}(n) = \max_{s \in n.unseen} q_{EB}(s)$$

Then, for SOC:

$$f_{SingletonEB}(n) = g(n) + h_{SingletonEB}(n) \quad (4)$$

For MKSP, we combine all costs directly into the f-value as follows. For each cell s, we define

$$f_{EB}(s) = \min_{\langle l_i, c_i \rangle \in n.L^+} q(s, l_i) + c_i$$

The MKSP can also be determined by a terminated agent:

$$f_{TER} = \max_{\langle l_i, c_i \rangle \in n.L^-} c_i$$

Then, for MKSP:

$$f_{SingletonEB}(n) = \max(\max_{s \in n.unseen} f_{EB}(s), f_{TER}) \quad (5)$$

$f_{SingletonEB}$ is equal to the cost of the longest (highest-cost) path among all paths of either terminated or non-terminated agents.

MTSP Heuristic. To adjust the MTSP heuristic for EB, G_{DLS} needs to consider the costs of the agents. Recall that, in G_{DLS}, there is an edge with cost 0 between V^* and each location of an agent. We split this edge into two directed edges. For each agent a_i in location l_i and cost c_i, we define a directed edge from V^* to the corresponding agent vertex with cost c_i, and a directed edge from the agent vertex to V^* with cost 0. This assures that the cost of each path (in MTSP) contains the cost of each agent arriving at that current node. Thus, the cost of the MTSP solution is not the h-value for n but is used as the f-value for n.

Duplicate Detection. Recall that for two nodes n_1 and n_2, without EB, we defined that n_1 can be discarded if three conditions are satisfied. For MKSP, g-values are not defined and condition 3 above ($g(n_2) \leq g(n_1)$) must be modified. For MKSP, node n_1 can be discarded if n_1 cannot lead to a solution that costs less than a solution reached from n_2, that is, if $\max_{\langle l_j, c_j \rangle \in n_2.L} c_j \leq \max_{\langle l_i, c_i \rangle \in n_1.L} c_i$ and $\forall \langle l_i, c_i \rangle \in n_1.L^+ \; \exists \langle l_j, c_j \rangle \in n_2.L^+$ s.t. $l_i = l_j$ and $c_j \leq c_i$.

Theorem 2 (Optimality). *MWRP-A* with EB obtains optimality.*

Proof outline. Assume a node n that represents a set of location-cost pairs for the agents that is part of the optimal solution to the problem. When node n is expanded, at least one of the agents must arrive at the closest location that sees new unseen cells. In EB, we consider the shortest path for each agent that leads to such a location. As we consider all possible combinations for such agent

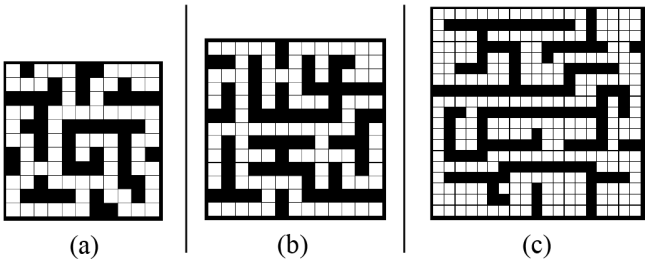

Figure 3: 11×11, 13×13 **grids and** 19×19 **grid maps**

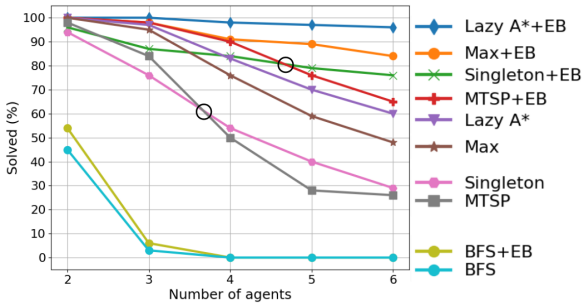

Figure 4: Success rate: 11×11 **grid (Figure 3(a)) for MKSP**

#Agents	Solved	h (+EB)	MKSP Cost	MKSP Exp.	MKSP Time	SOC Cost	SOC Exp.	SOC Time
2	94%	Singleton	30.1	6,901	38.6	52.9	21,714	47.4
		MTSP		108	16.1		387	6.6
		Max		**101**	15.5		**365**	6.8
		Lazy A*		**101**	**10.2**		366	**5.5**
3	74%	Singleton	19.1	2,124	26.9	43.6	23,731	55.3
		MTSP		51	20.3		432	9.8
		Max		**41**	19.1		**368**	9.9
		Lazy A*		**41**	**3.9**		**368**	**6.7**
4	66%	Singleton	14.9	790	26.2	36.1	17,646	57.7
		MTSP		94	50.9		632	19.7
		Max		**34**	35.6		**457**	18.0
		Lazy A*		**34**	**2.1**		**457**	**9.5**
5	61%	Singleton	12.2	493	23.4	30.5	11,009	59.4
		MTSP		73	48.0		334	16.7
		Max		**14**	26.8		234	16.8
		Lazy A*		**14**	**3.0**		**233**	**7.4**
6	47%	Singleton	10.1	299	17.8	23.7	3,664	60.5
		MTSP		99	52.9		114	12.6
		Max		**15**	34.0		**73**	12.5
		Lazy A*		18	**3.0**		**73**	**6.8**

Table 1: Results on the 11×11 **grid for MKSP and SOC**

movements, one of the newly generated nodes n' also represents a set of location-cost pairs for the agents that is part of the optimal solution. As the initial set of such pairs belongs to the optimal solution, inductively, the optimal solution must be reached.

□

8 EXPERIMENTAL RESULTS

We experimented with all our variants on a large number of test domains for both MKSP and SOC. Here, we report representative results (sometimes only for one cost function) that demonstrate the major trends that were also seen across the other non-reported experiments. Experiments were implemented in Python 3.8 and executed on an AMD EPYC 7702P 2.00GHz. MTSP was solved by CPLEX V:20.1.0.1.

8.1 Comparing Solvers

We first evaluate the following variants of MWRP-A* for minimizing MKSP, with and without EB: **(1)** Standard breadth-first search with no heuristic (denoted by BFS). **(2)** MWRP-A* with $h_{Singleton}$ (Singleton). **(3)** MWRP-A* with h_{MTSP} (MTSP). **(4)** The maximum between both heuristics (Max). **(5)** Lazy A* that first calculates $h_{Singleton}$ and, if needed, also calculates h_{MTSP} (Lazy A*). In our figures, we use "+EB" to denote the activation of EB.

We experimented on 100 problem instances for 2–6 agents whose start cells were randomly allocated on the 11×11 map presented in Figure 3(a), taken from `S20`.

Figure 4 shows the success rate (y-axis) under a time limit of 5 minutes for different numbers of agents (x-axis) for MKSP. As expected, for all solvers, increasing the number of agents decreases the success rate. Not using any heuristic (BFS), even with EB, performed poorly and the success rate dropped close to 0 already at 3 agents. Clearly, EB increased the success rate. In fact, without EB the success rate of all variants dropped to 60% or below at 6 agents, while all the EB variants were still above 65%. MTSP tends to outperform Singleton for a small number of agents but Singleton is better for more agents (the cross points are marked in the figure) due to the large overhead of calculating MTSP for many agents. Max had a better success rate than both heuristics alone, and Lazy A* was always the best. In fact, the success rate for Lazy A*+EB was very close to 100%. We also experimented with minimizing SOC on the same map, and similar trends were observed.

Table 1 presents the results, only for our solvers with EB and excluding BFS, for 2–6 agents (1st column). The percentage of instances that were solved by all solvers (MKSP and SOC) are presented in column 2. For these instances, we measured the average cost, #expansions, and time (in sec). For each heuristic (column 3), columns 4–6 present results for MKSP and columns 7–9 for SOC.

Naturally, the average cost for SOC is higher than that of MKSP because in SOC we sum up the costs of all paths while in MKSP the cost is determined by the longest path. As a result, the number of valid optimal solutions for MKSP is larger than that of SOC, i.e., in MKSP, the agents that do not have the longest path can have paths of different lengths. Consequently, all solvers performed more expansions in SOC than in MKSP. For minimizing MKSP, MTSP consumed larger runtime as the number of agents increased (due to its exponential overhead), while Singleton consumed shorter runtime as the number of agents increased. Hence, for more than 4 agents, Singleton outperforms MTSP, in terms of runtime. By contrast, for minimizing SOC, MTSP always outperforms Singleton. There are two reasons for this: (1) Singleton is less accurate when minimizing SOC, as it only estimates the cost of a single path. (2)

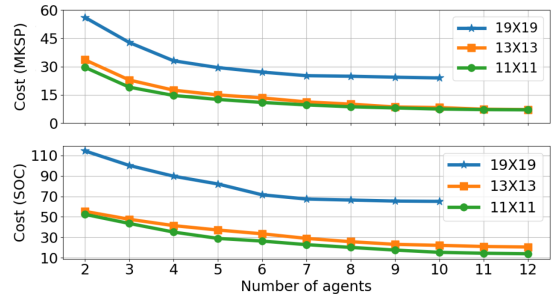

Figure 5: Increasing #agents on the various grids

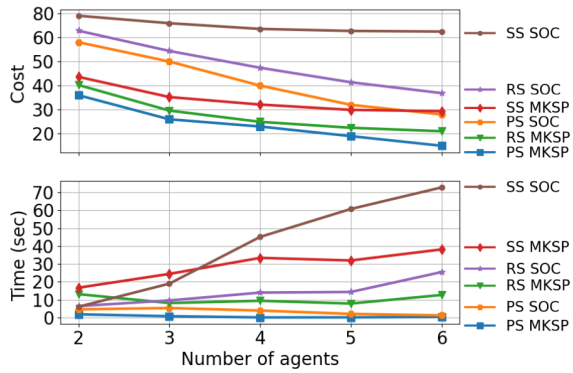

Figure 6: Comparison of SS, RS, and PS on the 11×11 grid

As shown by Kivelevitch et al. (2013), it is computationally harder for MTSP solvers that use constraint programming to minimize MKSP than minimizing SOC. Again, Lazy A* performed best. In general, there is a trade-off between the branching factor and the depth of the solution. More agents increases the branching factor but decreases the solution depth. We have experimented with other maps and while also varying the densities of obstacles and the same trends were observed.

To summarize, the best balance between simplicity and efficiency is achieved by Singleton, which is easy to implement and achieves a reasonable success rate. Lazy A* achieved the best performance, but is harder to implement. Next, as we consider other research questions, we only continue with our best solver, Lazy A*.

8.2 Varying the Number of Agents

Sometimes one can choose how many agents to employ. The aim is to minimize the number of agents while keeping the solution cost relatively low. We tested how the cost changes when the number of agents increases. Figure 5 shows the cost of the paths for MKSP (top) and SOC (bottom) for all maps of Figure 3. Adding more agents has a diminishing return in terms of the cost. Future work can explore the effect of additional features of the domain on the cost returned.

8.3 Analyzing Different Start Locations

In some cases, the locations from which the agents start their paths can be determined in advance. Thus, we evaluated on the grid of Figure 3(a) three policies for selecting the start locations: **(1)** Single start (SS), in which all agents start at the same location. **(2)** Random starts (RS), where every agent is randomly placed (as done in the experiments so far). **(3)** Pivots starts (PS), where the agents start by the method in which we select the locations of the pivots (as defined above). For SS, all 72 possible locations for placing all agents on the grid were tested. For RS, 100 random sets of start locations were tested (same instances as in previous experiments on Figure 3(a)). For PS, a single set of start locations was tested. Only our best solver Lazy A* was considered and it solved more than 86% of the instances (5 minutes time limit) for each number of agents (2–6), for both MKSP and SOC.

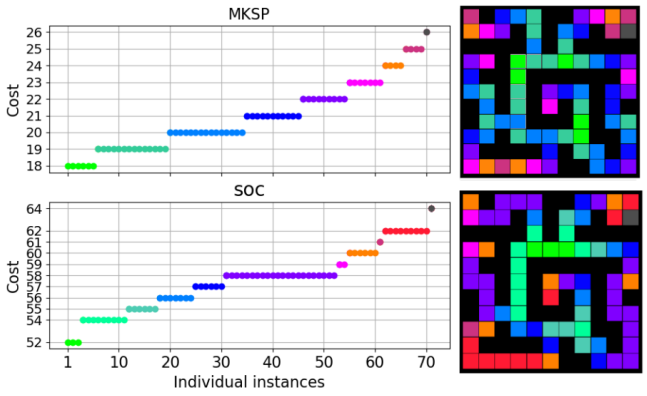

Figure 7: Cost of SS, on the 11×11 grid

For the solved instances, Figure 6 presents the average solution cost (top) and the average time (bottom). Finding SOC solutions consumed more time than MKSP, and solving SS instances consumed more time than solving RS instances which itself consumed more time than solving the PS instance. Moreover, the cost of SOC is higher than the cost of MKSP, and the cost of SS is higher than the cost of RS, which is higher than PS. While RS and PS scatter the agents across the map, SS forces them to share a start location. Thus, in SS, the paths of the agents are longer. We also compared the lowest cost solution found by each policy for both MKSP and SOC (instead of average). Here also, PS was always the best, then RS and then SS. Thus, for obtaining a solution of low cost, we suggest using PS. It is for future work to suggest additional policies for selecting start locations for the agents.

Experiments with SS. In some cases, all agents must start at the same location, e.g., at a charging area. To understand which locations are better for SS, we iterated over all 72 locations and placed 4 agents in the same location. Figure 7 (left) shows the cost for these instances in increasing order for MKSP and SOC. The difference between the best and worst cases is $\approx$ 20% for SOC and $\approx$ 30% for MKSP. In order to save this margin, one might decide to find the best location to start. Figure 7 (right) shows the locations of these instances colored according to their costs. Between two neighboring cells, the cost can change (at most) by m for SOC and by 1 for MKSP. However, in SOC, cells are highly correlated with their neighbors. For example, all three optimal locations are adjacent to each other. For MKSP, this correlation is less evident and colors are more scattered. Naturally, cells in the center achieved lower costs than cells on the border.

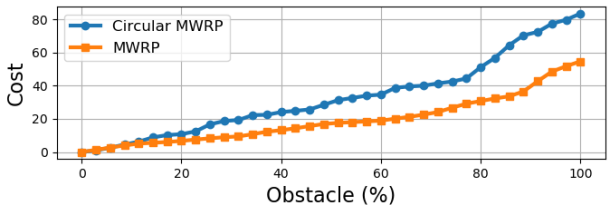

Figure 8: Varying obstacle percentage: 13×13 grid for Circular MWRP and (non-circular) MWRP solutions, for SOC.

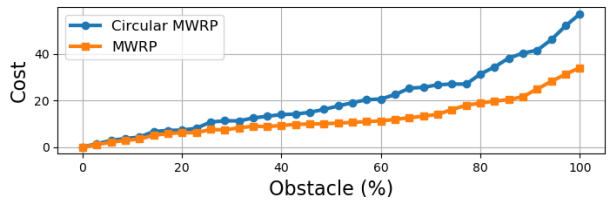

Figure 9: Varying obstacle percentage: 13×13 grid for Circular MWRP and (non-circular) MWRP solutions, for MKSP.

#Agents	Circular MWRP	MWRP	Ratio
1	98.3	71.4	1.38
2	88.6	55.7	1.59
3	82.0	48.4	1.69
4	73.5	40.7	1.81

Table 2: Varying #agents: 13×13 grid for Circular MWRP and (non-circular) MWRP solutions, for minimizing SOC.

#Agents	Circular MWRP	MWRP	Ratio
1	98.3	71.4	1.38
2	58.0	34.6	1.67
3	38.3	22.4	1.71
4	30.0	16.9	1.78

Table 3: Varying #agents: 13×13 grid for Circular MWRP and (non-circular) MWRP solutions, for minimizing MKSP.

This is unfortunate since, for example, in buildings, natural start locations (doors) are on the border. By contrast, elevators tend to be more at the center of the floor.

8.4 Circular Variants of MWRP

As mentioned, we focused on a variant of MWRP in which the agents can finish their paths in any location. However, in some realistic cases the agents must return to their corresponding start locations and circular paths should be returned. Therefore, we explain how to find optimal circular paths, and compare the two variants of the problem, namely, circular and non-circular MWRP. Above, we modeled MWRP as a search problem. To model Circular-MWRP as a search problem, two definitions are modified, as follows.

Goal Node. In MWRP, a goal node is determined when all cells are seen. In Circular-MWRP, a node is goal iff all cells are seen and also all agents are at their start cells.

Successor Function. In MWRP, each agent can either move to a neighboring cell or perform a terminate action. In Circular-MWRP, each agent can either move to a neighboring cell or return to its start cell (via the shortest path from its current cell) and terminate.

Naturally, an optimal circular solution always has a higher cost than an optimal non-circular solution.[3] Given a non-circular solution, each agent can return to its start cell through the same path it performed. Thus, in the worst case, an optimal circular solution costs twice than an optimal non-circular one. Nevertheless, we are interested to explore the differences between the costs of optimal solutions to the two problems. For this we do not have to modify our heuristics but can use them as is because they are also admissible heuristics for the circular variant of the problem. Future work can improve our heuristics for other variants as well.

To examine the effect of varying the number of obstacles on our solvers, we performed a similar experiment on the 13×13 grid map as done by `S20` for a single agent. Iteratively, we randomly removed two obstacles from the grid, resulting in a new grid with fewer obstacles. For each such percentage of obstacles, we created 50 instances of three randomly placed agents and measured the average cost of circular and non-circular solutions. Figures 8 and 9 show the result of this experiment for minimizing SOC and MKSP, respectively. As expected, the cost of circular solutions was no more than twice the cost of non-circular solutions. Also, increasing the number of obstacles increases the difference between the cost of the two solutions. When more obstacles exist, some agents must reach farther cells which also increases the cost of their return.

We also experimented on the full 13×13 grid while varying the number of agents (1–4). The results are shown in Tables 2 and 3 for SOC and MKSP, respectively. Here, we also present the ratio between the cost of both solutions. Interestingly, as the number of agents increases, the ratio gets closer to two, i.e., the circular solution worsens compared to the non-circular one. When more agents exist, in MWRP, some agents can be sent directly to distant locations and terminate there. With fewer agents, in both standard- and Circular-MWRP solutions, each agent will have to move around the map to see a relatively large number of locations and the difference between the costs of the two versions will not be large.

9 CONCLUSIONS AND FUTURE WORK

We generalized WRP for multiple agents (MWRP). We suggested two heuristics for MWRP: $h_{Singleton}$ and h_{MTSP}. We proposed EB, which enhances the search. Our experiments show that Lazy A*, which combines both heuristics, performs best. We also observed that selecting the pivots locations for start cells is the best policy in terms of cost. Future work can (1) propose other heuristics for MWRP. (2) Develop decoupled approaches for solving MWRP, which search for paths for each agent separately. (3) Suggest additional policies for selecting locations for the agents. (4) Consider the case of avoiding collisions between agents.

[3] Note that deleting the last edge of an optimal circular path results in a non-circular path, but not necessarily in an optimal one.

ACKNOWLEDGEMENTS

This research was sponsored by the United States-Israel Binational Science Foundation (BSF) under grant numbers 2017692 and 2021643, and by Israel Science Foundation (ISF) under grant number 844/17.

REFERENCES

[1] Josep Aulinas, Yvan Petillot, Joaquim Salvi, and Xavier Lladó. 2008. The SLAM problem: a survey. *Artificial Intelligence Research and Development* (2008), 363–371.
[2] Alex Bavelas. 1950. Communication Patterns in Task-Oriented Groups. *The Journal of the Acoustical Society of America* 22, 6 (1950), 725–730.
[3] Jack E Bresenham. 1965. Algorithm for computer control of a digital plotter. *IBM Systems journal* 4, 1 (1965), 25–30.
[4] Svante Carlsson, Bengt J Nilsson, and Simeon Ntafos. 1993. Optimum guard covers and m-watchmen routes for restricted polygons. *International Journal of Computational Geometry & Applications* 3, 01 (1993), 85–105.
[5] Wei-Pang Chin and Simeon Ntafos. 1986. Optimum watchman routes. In *the Second Annual Symposium on Computational Geometry*. 24–33.
[6] Pooyan Fazli, Alireza Davoodi, and Alan K Mackworth. 2013. Multi-robot repeated area coverage. *Autonomous Robots* 34, 4 (2013), 251–276.
[7] Mengyu Fu, Alan Kuntz, Oren Salzman, and Ron Alterovitz. 2019. Toward asymptotically-optimal inspection planning via efficient near-optimal graph search. *Robotics Science and Systems: Online Proceedings* (2019).
[8] Michael R. Garey and David S. Johnson. 1979. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., New York, NY, USA.
[9] Elad Kivelevitch, Kelly Cohen, and Manish Kumar. 2013. A Binary Programming Solution to the Min-Max Multiple-Depots, Multiple Traveling Salesman Problem. In *AIAA Infotech at Aerospace Conference*. 1–11.
[10] Eli Packer. 2008. Computing Multiple Watchman Routes. In *the International Workshop on Experimental and Efficient Algorithms*. Springer, 114–128.
[11] Nicolas Rivera, Carlos Hernández, and Jorge A. Baier. 2017. Grid Pathfinding on the $2k$ Neighborhoods. In *the AAAI Conference on Artificial Intelligence (AAAI)*. 891–897.
[12] Shawn Seiref, Tamir Jaffey, Margarita Lopatin, and Ariel Felner. 2020. Solving the watchman route problem on a grid with heuristic search. In *the International Conference on Automated Planning and Scheduling (ICAPS)*. 249–257.
[13] Shawn Skyler, Dor Atzmon, Tamir Yaffe, and Ariel Felner. 2022. Solving the Watchman Route Problem with Heuristic Search. *Journal of Artificial Intelligence Research (JAIR)* 75 (2022), 747–793.
[14] Samerkae Somhom, Abdolhamid Modares, and Takao Enkawa. 1997. A self-organising model for the travelling salesman problem. *Journal of the Operational Research Society* 48, 9 (1997), 919–928.
[15] Roni Stern, Nathan R. Sturtevant, Ariel Felner, Sven Koenig, Hang Ma, Thayne T. Walker, Jiaoyang Li, Dor Atzmon, Liron Cohen, T. K. Satish Kumar, Roman Barták, and Eli Boyarski. 2019. Multi-Agent Pathfinding: Definitions, Variants, and Benchmarks. In *the International Symposium on Combinatorial Search (SoCS)*. 151–159.
[16] Takafumi Taketomi, Hideaki Uchiyama, and Sei Ikeda. 2017. Visual SLAM algorithms: a survey from 2010 to 2016. *IPSJ Transactions on Computer Vision and Applications* 9 (2017), 1–11.
[17] David Tolpin, Tal Beja, Solomon Eyal Shimony, Ariel Felner, and Erez Karpas. 2013. Toward rational deployment of multiple heuristics in A*. In *the International Joint Conference on Artificial Intelligence (IJCAI)*. 674–680.
[18] Tamir Yaffe, Shawn Skyler, and Ariel Felner. 2021. Suboptimally Solving the Watchman Route Problem on a Grid with Heuristic Search. In *the International Symposium on Combinatorial Search (SoCS)*. 106–114.