Proceedings of the Thirty-Sixth International Conference on Automated Planning and Scheduling (ICAPS 2026)

Scalable Algorithms with Provable Optimality Bounds for the Multiple Watchman Route Problem

Srikar Gouru[1], **Ariel Felner**[2], **Jiaoyang Li**[1]
[1]Carnegie Mellon University
[2]Ben-Gurion University of the Negev
srikargouru@cmu.edu, felner@bgu.ac.il, jiaoyangli@cmu.edu

Abstract

In this paper, we tackle the Multiple Watchman Route Problem (MWRP), which aims to find a set of paths that M watchmen can follow such that every location on the map can be seen by at least one watchman. First, we propose multiple methods to reduce the state space over which a search needs to be conducted by pruning map areas that are guaranteed to be seen en route to other areas. Next, we introduce MWRP-CP^3, an efficient optimal planner that combines these methods with techniques that improve the quality and calculation time of existing heuristics. We present several suboptimal algorithms with bounds on solution quality, including MxWA*, a general variant of weighted A* for makespan problems. We also present anytime variations of our suboptimal algorithms, as well as techniques to improve an existing suboptimal solution by solving multiple decomposed sub-problems. We show that MWRP-CP^3 can reduce the search space by more than 95% and runs more than 200$\times$ faster than existing optimal algorithms on 2D grid maps. We also show that our suboptimal algorithms solve maps 3$\times$ larger than those solvable by MWRP-CP^3. See `mwrp-cp3.github.io` for the open source codebase and video demonstrations.

1 Introduction

Coverage Path Planning (CPP) is the task of computing a path that passes over every point in a given environment. CPP plays a vital role in emergency scenarios such as searching for survivors in disaster relief (Cho et al. 2021) and locating active wildfires (Bezas et al. 2022). Extensive research has been conducted for exploration in *unknown* environments including frontier-based (Yamauchi 1997) and sampling-based (Steinbrink et al. 2021) algorithms. For CPP in *known* environments, methods such as cellular decomposition (Huang 2001) and genetic algorithms (Jimenez et al. 2007) have been studied to compute optimal paths. However, these methods are designed for robots with a pre-determined coverage radius, such as lawnmowers and vacuums, rather than robots whose coverage may instead depend on visibility or line-of-sight.

Specifically, we focus on the *Watchman Route Problem* (WRP) which requires a traveling agent, the *watchman*, to find the shortest path such that every point on the environment can be seen from at least one location on the path. WRP was first introduced by Chin and Ntafos, who presented an efficient solution for rectilinear polygons and proved that the problem was NP-hard in polygons with holes (pang Chin and Ntafos 1988). Since then, many problem variations have been introduced. The k-Watchman Route Problem (k-WRP) (Packer 2008) involves k traveling watchmen whose combined vision must encompass the entire environment, with the goal of minimizing either the longest watchman's path (*min-max / makespan*) or the total path length (*min-sum*). Computing optimal paths for an arbitrary k watchmen was proven to be NP-hard for both the makespan (Mitchell and Wynters 1991) and min-sum (Nilsson 1995) objectives in simple polygons. Moreover, WRP can be either *anchored*, in which the watchmen have predetermined starting locations, or *floating*, in which the watchman paths can start anywhere.

Recently, a discrete variation of single-watchman WRP was introduced for arbitrary graphs, with a focus on 2D grid maps. Unlike the classic WRP and k-WRP problems, graph-based WRP works on an arbitrary *Line-Of-Sight* (LOS) function that determines which cells are visible from a given cell (Seiref et al. 2020). Additionally, graph-based WRP makes no prior assumptions on the environment geometry and the watchman's LOS, allowing for complex problem instances to be modeled. The problem was proven to be NP-hard to solve optimally. Suboptimal algorithms have also been proposed (Yaffe, Skyler, and Felner 2021).

Graph-based WRP has since been extended to the graph-based *Multiple Watchman Route Problem* (MWRP), in which every cell must be visible from at least one cell along at least one of the watchman paths (Livne et al. 2023). A new variant of A* called MWRP-A* was introduced, which performs a joint-space heuristic search to compute an optimal set of paths for both the min-sum and makespan objectives. They were able to solve grids of up to 200 free cells and up to 6 agents in several seconds. However, MWRP-A* is infeasible for real-world scenarios where maps consist of thousands of cells. We focus on the makespan objective since most exploration problems are time-critical due to some emergency (i.e., disaster relief, firefighting). Our contributions are summarized as follows:

1. **Optimal Search Algorithm:** We present MWRP-CP^3, which consists of methods for reducing the state space of the algorithm by determining that some cells in the map are strictly harder to see than others. Additionally,

MWRP-CP^3 includes algorithmic enhancements that significantly improve heuristic quality and calculation time.

2. **Bounded Suboptimal Search (BSS):** We introduce *Minimax Weighted A** (MxWA*), a general variant of weighted A* (Pohl 1970) that efficiently computes bounded-quality solutions for makespan problems. Additionally, we introduce two heuristics that are used in conjunction with Focal Search (Pearl and Kim 1982) to solve the MWRP problem with makespan. We also formulate anytime variations for both algorithms.
3. **Postprocessing Framework:** We present a framework that efficiently improves the makespan of an existing suboptimal solution by partitioning the map into and solving fast, decomposed sub-problems for each agent.

We show that MWRP-CP^3 can reduce the search space by more than 95% on complex maps and computes optimal paths more than 200× faster than existing baselines. We also show that our suboptimal algorithms scale in terms of map size (1,500+ grid cells) and agent count (5+ agents).

2 Background

The WRP and MWRP problems can be represented as any graph structure (Seiref et al. 2020; Livne et al. 2023), and all of our techniques and algorithms are generalizable to arbitrary graphs. For simplicity, we formulate the problem as a 2D grid graph composed of free and obstacle cells, a common approach in coverage-related problems.

2.1 Problem Formulation

The MWRP problem is defined by the tuple $(M, \mathcal{C}, \mathcal{U}, \mathcal{S}, \mathcal{N}, \mathcal{L}, \mathcal{O})$, where M is the number of agents, $\mathcal{C}$ is the set of free cells on the grid that the agents can be at, $\mathcal{U} \subseteq \mathcal{C}$ is the set of cells that need to be seen by at least one of the agents, and $\mathcal{S} = \{\mathcal{S}_1, ..., \mathcal{S}_M\}$, where $\mathcal{S}_k \in \mathcal{C}$ is the starting location of agent a_k. In general, $\mathcal{U} = \mathcal{C}$, however we separate these terms because we introduce certain subproblems where not every cell in $\mathcal{C}$ needs to be explicitly seen. The neighbor function, $\mathcal{N}$, defines the cells that an agent can move to in a single action from a given cell, and the LOS function, $\mathcal{L}$, defines all cells visible to an agent from a given cell. Figure 1 (left) visualizes *four-way movement*, an example neighbor function, and *Bresenham LOS*, an example LOS function commonly used in computer graphics that simulates a continuous field-of-view in discrete maps (Bresenham 1965). We define the *watchers* function, $\mathcal{W}$, as the inverse of the LOS function, such that $s_2 \in \mathcal{L}(s_1) \leftrightarrow s_1 \in \mathcal{W}(s_2) \, \forall \, s_1, s_2 \in \mathcal{C}$. For many LOS functions, including Bresenham LOS, $\mathcal{W}$ may not be symmetric, meaning there is no guarantee that $\mathcal{W}(s) = \mathcal{L}(s)$. We define $d(s_1, s_2)$ as the length of the shortest traversable path between s_1 and s_2.

A solution to the MWRP problem must be a set of paths $\boldsymbol{\pi} = \{\pi_1, ..., \pi_M\}$, where each path π_k is a list of cells that agent a_k traverses starting at $\mathcal{S}_k$ and following the movement constraints defined by $\mathcal{N}$. A set of paths, $\boldsymbol{\pi}$, is a solution to the problem if every cell in $\mathcal{U}$ is seen during at least one agent's path. That is, defining the function $\mathcal{P}(\boldsymbol{\pi}) = \bigcup \pi_k$, $\boldsymbol{\pi}$ is a solution if $\mathcal{W}(s) \cap \mathcal{P}(\boldsymbol{\pi}) \neq \emptyset$ for every cell $s \in \mathcal{U}$. The cost of a path is its length, defined as $c(\pi_k) = |\pi_k|$. $\mathcal{O}$ is the *makespan* objective function, defined as $\mathcal{O}(\boldsymbol{\pi}) = \max_{\pi_k \in \boldsymbol{\pi}} c(\pi_k)$.

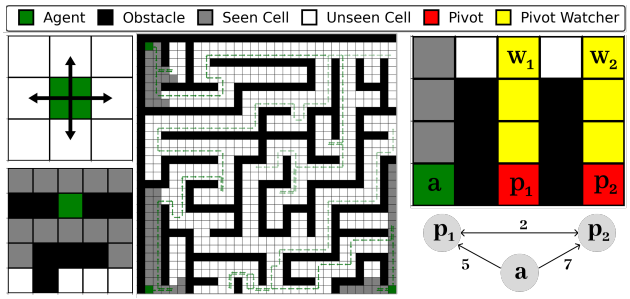


Figure 1: Examples of four-way movement (top left), Bresenham LOS (bottom left), an MWRP problem with the optimal makespan solution (middle), an example scenario for the mTSP heuristic (top right), and its corresponding G_{DLS} (bottom right). The legend applies to all figures in this paper.

Because exploration problems generally require few agents searching over a large area, we assume that each cell $s \in \mathcal{C}$ can hold all M agents, and thus we do not consider agent-agent collisions for our solution. This assumption was also used in previous work (Livne et al. 2023). The problem is solved once offline, and we assume perfect information regarding the map structure and agent starting locations. Figure 1 (middle) shows an example problem and a makespan-optimal solution on a 32×32 Maze map with 3 agents.

2.2 MWRP-A* Algorithm

In this section we summarize the MWRP-A* algorithm by Livne et al., a variation of joint-space A* that optimally solves the MWRP problem (Livne et al. 2023). **Node**: Each node in the search tree consists of a list of tuples, (s_k, c_k), representing the current location and the cumulative cost for agent a_k. Additionally, each node keeps track of its *residual set* $R \subseteq \mathcal{U}$, which is the set of residual (or remaining) cells that need to be seen. The cost of the node is $\max_k\{c_k\}$. **Root node**: The tuples are $(\mathcal{S}_k, 0)$ for each agent a_k, and the residual set, $R = \mathcal{U} \setminus \bigcup \mathcal{L}(S_k)$, represents the residual cells not visible from any agent's starting location. **Goal node:** Any node where $R = \emptyset$ is considered a goal node.

Successors Successor nodes are generated using a joint-space expansion. To avoid generating nodes where agents do not see any new cells, MWRP-A* utilizes the *Expanding Borders* mechanism, which finds nearby cells for each agent that have LOS to a previously unseen residual cell. These new cells, along with the cumulative cost to reach them, are the successor states of the agent. Each agent also has the option to *terminate*, which removes it from the set of agents in the successor node. The successor nodes are the Cartesian product of the successor states for each agent. In each successor node, R is computed by copying R from the parent node and removing cells that the agents have LOS to from their new locations.

Node Dominance To avoid unnecessary re-expansions, *dominated* nodes are pruned out during the search. Node n *dominates* node n' if (1) all of their agent locations are equal ($n.s_k = n'.s_k$), (2) the residual set for node n is a subset of the residual set for node n' ($n.R \subseteq n'.R$), and (3) the cumulative cost for each agent in node n is less than or equal to the cumulative cost in node n' ($n.c_k \leq n'.c_k$).

Lazy Heuristic Evaluation Nodes are initially inserted into the *OPEN* list with their f-value calculated using the *Singleton heuristic*. If a node is popped from *OPEN* for the first time, a more informed f-value is calculated using the *multiple Traveling Salesman Problem (mTSP) heuristic*, and the node is re-inserted into *OPEN* with its new f-value. When a node is popped from *OPEN* for the second time, it is expanded and successors are generated.

Singleton Heuristic The Singleton heuristic defines $f_s = \min_k\{c_k + w_{s,k}\}$ as the lowest agent cost at which cell s can be seen, where $w_{s,k}$ is the minimum cost for agent a_k to reach a watcher of s. Since every cell in R must be seen, the admissible Singleton f-value is $\max\{f_s\}$ among all $s \in R$.

mTSP Heuristic First, a set of *pivots* is selected from R such that no two pivots share any watchers. Next, a graph G_{DLS} is constructed, where each vertex v_p in G_{DLS} corresponds to pivot p, and each vertex v_a corresponds to agent a. Since any solution requires all pivots to be seen by at least one agent, the heuristic is computed by solving mTSP on the graph G_{DLS}, where each vertex v_p must be reached by at least one agent, with the agents starting at v_a. For the heuristic to be admissible, edges in G_{DLS} are calculated with underestimated costs. Edge costs from agent vertices v_a to pivot vertices v_p are calculated as the minimum distance between agent a and any watcher of pivot $w \in \mathcal{W}(p)$. Edge costs from pivot vertices v_{p_i} to other pivot vertices v_{p_j} are calculated as the minimum distance between any watcher $w_i \in \mathcal{W}(p_i)$ and any watcher $w_j \in \mathcal{W}(p_j)$, as a lower bound on the travel distance to go from seeing pivot p_i to pivot p_j.

Figure 1 (top right) shows an example search node with a single agent a, pivots p_1 and p_2, and the corresponding G_{DLS} (bottom right). The closest distance from agent a to a watcher of p_1 (labeled w_1) is 5, and the closest distance from agent a to a watcher of p_2 (labeled w_2) is 7. Additionally, the closest distance from any watcher of p_1 to any watcher of p_2 is 2 (the distance from w_1 to w_2), thus resulting in the G_{DLS} graph shown in Figure 1 (bottom right).

Defining h_k as the path length of each agent a_k on G_{DLS}, solving mTSP to minimize $\min\{c_k + h_k\}$ (Gavish 1976) results in an admissible f-value. However, this f-value is not consistent, so we utilize the *pathmax* technique to ensure consistency by setting $f(n) = \max\{f(n), f(n_p)\}$, where n_p is node n's parent during the A* search (Mero 1984).

3 MWRP-CP^3

This section describes our optimal search algorithm, MWRP-CP^3, which is MWRP-A* with Cell and Path dominance, Pivot pruning, and Parallel heuristic calculation. We first introduce cell and path dominance, two methods for reducing the algorithm's state space. Then, we introduce pivot pruning and parallel heuristic calculation, two enhancements that directly speed up the original MWRP-A* algorithm.

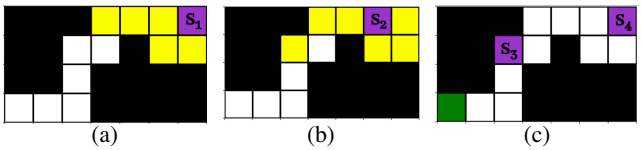


Figure 2: Figures(a) and (b) show the watchers (yellow) of cells s_1 and s_2 (purple), respectively. The cells are also watchers of themselves. Figure(c) shows an example with an agent (green) where s_3 is along the agent's path to s_4.

Algorithm 1: Cell Dominance (CD)

Input: $\mathcal{W}, \mathcal{U}$
Output: Updated $\mathcal{U}$ with dominated cells pruned

1: **for** $s_i \in \mathcal{U}$ **do**
2: **for** $s_j \in \mathcal{U} \setminus \{s_i\}$ **do**
3: **if** $\mathcal{W}(s_i) \subseteq \mathcal{W}(s_j)$ **then** $\mathcal{U} = \mathcal{U} \setminus \{s_j\}$;
4: **return** $\mathcal{U}$;

3.1 State Space Reduction

Each node in the search space is uniquely defined by the location of each agent, which is a cell in $\mathcal{C}$, and the residual set R, which is a subset of $\mathcal{U}$. Thus, the state space is $O(|\mathcal{C}|^M 2^{|\mathcal{U}|})$. We introduce *cell* and *path dominance*, two methods to reduce $\mathcal{U}$ by identifying cells that can be ignored during the search, a process that we call *cell pruning*. We prove that both methods do not alter the set of possible solutions, thus preserving the algorithm's optimality guarantees.

Cell Dominance The first method, *cell dominance* (CD), stems from the observation that some cells are guaranteed to be seen when an agent sees other cells, no matter where the agent is. Intuitively, looking at the example shown in Figure 2(a) and (b), it is impossible to see s_1 without also seeing s_2. Formally, this is because the watchers of s_1 are a subset of the watchers of s_2. Thus, no matter what watcher of s_1 an agent sees the cell from, it simultaneously sees s_2. We refer to this property as cell dominance, specifically that s_1 *dominates* s_2. During our search, we ignore s_2, since seeing s_1 must also result in seeing s_2. Theorem 1 proves that cell dominance does not sacrifice optimality.

Theorem 1. *Given a cell $s_j \in \mathcal{U}$, if there exists another cell $s_i \in \mathcal{U}$ such that $\mathcal{W}(s_i) \subseteq \mathcal{W}(s_j)$, then any solution $\boldsymbol{\pi}$ that sees $\mathcal{U} \setminus \{s_j\}$ also sees $\mathcal{U}$.*

Proof. Let $\boldsymbol{\pi}$ be any solution that sees every cell in $\mathcal{U} \setminus \{s_j\}$. Since $s_i \in \mathcal{U} \setminus \{s_j\}$, the solution must include a watcher of s_i, so there exists a cell $k \in \mathcal{P}(\boldsymbol{\pi}) \cap \mathcal{W}(s_i)$. Since $\mathcal{W}(s_i) \subseteq \mathcal{W}(s_j)$, it must be that $k \in \mathcal{W}(s_j)$, thus $\boldsymbol{\pi}$ sees s_j. □

In Algorithm 1, we loop through every pair of unseen cells (lines 1-2) and prune one of them if it is dominated by the other (line 3). Since each cell has at most $|\mathcal{C}|$ watchers, the runtime complexity of CD is $O(|\mathcal{C}||\mathcal{U}|^2)$. CD is executed

Algorithm 2: subgraphBFS

Input: $\mathcal{S}, \mathcal{N}, \mathcal{C}'$
Output: $\mathbf{\Pi}$ containing all reachable cells within $\mathcal{C}'$

1: $Q = \mathcal{S}$;
2: $\mathbf{\Pi} = \emptyset$;
3: **while** $Q \neq \emptyset$ **do**
4: $\quad s = Q.pop()$;
5: $\quad$ **if** $s \in \mathbf{\Pi}$ ***or*** $s \notin \mathcal{C}'$ **then continue**;
6: $\quad \mathbf{\Pi} = \mathbf{\Pi} \cup s$;
7: $\quad$ **for** $n \in \mathcal{N}(s)$ **do** $Q = Q \cup n$;
8: **return** $\mathbf{\Pi}$;

Algorithm 3: Path Dominance (PD)

Input: $\mathcal{W}, \mathcal{N}, \mathcal{U}, \mathcal{S}$
Output: Updated $\mathcal{U}$ with dominated cells pruned

1: **for** $s_j \in \mathcal{U}$ **do**
2: $\quad \mathcal{C}' = \mathcal{C} \setminus \mathcal{W}(s_j)$;
3: $\quad \mathbf{\Pi} =$ `subgraphBFS` $(\mathcal{S}, \mathcal{N}, \mathcal{C}')$;
4: $\quad$ **for** $s_i \in \mathcal{U} \setminus \{s_j\}$ **do**
5: $\quad\quad$ **if** $\mathcal{W}(s_i) \cap \mathbf{\Pi} = \emptyset$ **then**
6: $\quad\quad\quad \mathcal{U} = \mathcal{U} \setminus \{s_j\}$;
7: $\quad\quad\quad$ **break**;
8: **return** $\mathcal{U}$;

once per problem instance to compute the updated $\mathcal{U}$ prior to running the A* algorithm discussed in Section 2.2.

Path Dominance The second method, *path dominance* (PD), stems from the idea that given the starting locations of the agents, the agents are guaranteed to see some cells on the way to seeing other cells. In the example shown in Figure 2(c), s_3 lies on the agent's path to seeing s_4, and it is impossible to see s_4 without seeing s_3 first. Thus, for any pair of distinct cells s_i and s_j in $\mathcal{U}$, if all the paths to seeing s_i require the agent to also see s_j, then we can safely ignore s_j during our search, since any solution that sees s_i is guaranteed to also see s_j. We call this behavior *path dominance*, specifically that s_i *dominates* s_j. For a cell $s_j \in \mathcal{U}$, we determine if another cell $s_i \in \mathcal{U}$ dominates it by performing a multi-source breadth-first search (BFS), starting at $\mathcal{S}$, to find all cells $\mathbf{\Pi}$ that can be reached by at least one agent without seeing s_j. If there is some cell s_i such that no watcher of s_i is in $\mathbf{\Pi}$, then s_i *dominates* s_j. Theorem 2 proves that path dominance does not sacrifice optimality.

Theorem 2. *Given a cell $s_j \in \mathcal{U}$, we define Π as the set of all paths that originate at the starting location of an agent and do not see s_j from any cell along the path. We also define $\mathbf{\Pi} = \bigcup_{\pi_i \in \Pi} \pi_i$ as the set of cells that is in at least one path in Π. If there exists a cell $s_i \neq s_j$ such that $\mathcal{W}(s_i) \cap \mathbf{\Pi} = \emptyset$, then any solution $\boldsymbol{\pi}$ that sees $\mathcal{U} \setminus \{s_j\}$ also sees $\mathcal{U}$.*

Proof. Let $\boldsymbol{\pi}$ be any solution that sees every cell in $\mathcal{U} \setminus \{s_j\}$. Since $s_i \in \mathcal{U} \setminus \{s_j\}$, there exists a watcher $k \in \mathcal{W}(s_i)$ that was visited by at least one agent, so $k \in \pi_a$ for some $\pi_a \in \boldsymbol{\pi}$. Since $\mathcal{W}(s_i) \cap \mathbf{\Pi} = \emptyset$, we know $k \notin \mathbf{\Pi}$. This means that $\pi_a \notin \Pi$, so π_a must have seen s_j along its path. □

First, we introduce `subgraphBFS` (Algorithm 2), a modified floodfill algorithm that only expands cells within a given cell set $\mathcal{C}'$ (line 5). The method returns all cells in $\mathcal{C}'$ reachable from $\mathcal{S}$ and has a runtime complexity of $O(|\mathcal{C}'|)$.

In Algorithm 3 we describe the PD algorithm, in which we loop through all cells $s_j \in \mathcal{U}$ to determine if s_j is path dominated by any other cell s_i. First, we compute $\mathcal{C}'$, the subset of $\mathcal{C}$ with all watchers of s_j removed (line 2). Then we call `subgraphBFS` on $\mathcal{C}'$ to compute $\mathbf{\Pi}$, the set of cells that an agent can reach while avoiding LOS to s_j (line 3). If there is a cell s_i that has no watchers in $\mathbf{\Pi}$, then s_i dominates s_j, so s_j is pruned (lines 4-6). The runtime complexity for this check is $O(|\mathcal{C}||\mathcal{U}|)$, which dominates the $O(|\mathcal{C}|)$ runtime of the *subgraphBFS* algorithm. Thus, like CD, the runtime complexity of the PD algorithm is $O(|\mathcal{C}||\mathcal{U}|^2)$. PD is run once per problem instance to compute the updated $\mathcal{U}$ prior to running the A* algorithm discussed in Section 2.2.

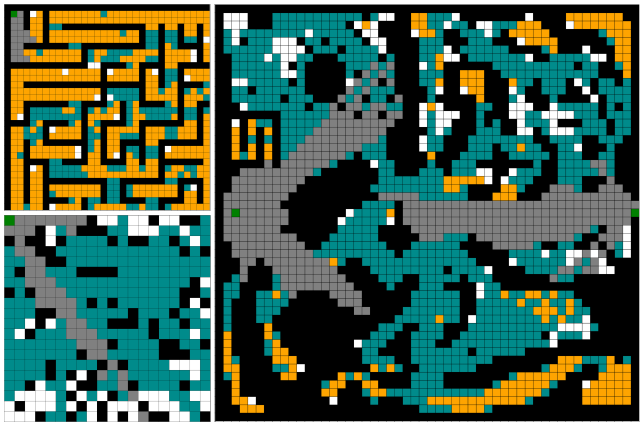


Figure 3: Comparison of state space reduction showing cells removed by *cell dominance* (orange only) and *path dominance* (orange + blue) on a Maze map (top left) (Stern et al. 2019), a randomized map (bottom left), and a Minecraft-inspired map (right) (Chong, Li, and Sycara 2024).

We also note that the set of cells pruned by CD is a subset of those pruned by PD. If s_i dominates s_j in CD, then there exists no cell from which we can see s_i without also seeing s_j, thus s_i dominates s_j in PD as well. Figure 3 visualizes the cells dominated by CD and PD. Our experiments show that CD generally runs faster than PD due to time-intensive BFS computation in PD. Thus, we first run CD followed by PD, reducing the cells that PD needs to loop through. We call this method cell and path dominance (CPD).

3.2 Pivot Pruning

Next, we describe an algorithmic enhancement to MWRP-A*. When computing the mTSP heuristic, we know that any set of pivots, where no two pivots share watchers, results in an admissible mTSP heuristic. MWRP-A* greedily uses as many pivots as possible. However, consider the example shown in Figure 4(a). From agent a, the closest watcher of p_1 is w_1, 15 cells away, and the closest watcher of p_2 is w_3 5 cells away. Additionally, the closest watchers between p_1

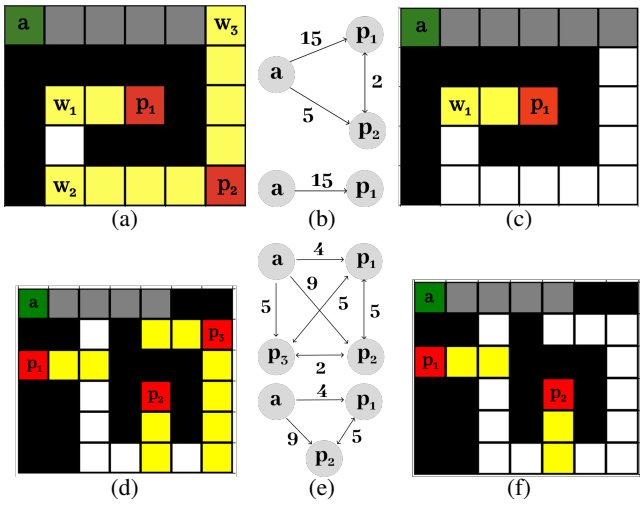


Figure 4: Examples where pivot pruning improves (top) and worsens (bottom) the heuristic. In Figures(b) and (e), the top graph corresponds to the left Figures(a/d), and the bottom graph corresponds to right Figures(c/f).

and p_2, w_1 and w_2, are 2 cells apart. This results in the graph G_{DLS} shown in Figure 4(b) (top graph). The mTSP heuristic solved on G_{DLS} results in the path $a \rightarrow p_2 \rightarrow p_1$, with a heuristic value of $5 + 2 = 7$. However, if we remove p_2 (Figure 4(c)), then the only edge is $a \rightarrow p_1$ (Figure 4(b), bottom graph), and solving mTSP gives a heuristic value of 15. We call this behavior *shortcut removal*, since p_2 provides a shortcut between a and p_1. To maximize the admissible heuristic value, we greedily remove pivots that may provide shortcuts in the mTSP solution.

The shortcut that pivot p_i provides between agent a_k and pivot p_j is $s(p_i, p_j, a_k) = e(a_k, p_j) - (e(a_k, p_i) + e(p_i, p_j))$, where e is the distance between two vertices in G_{DLS}. We use a while loop to repeatedly prune out the pivot that provides the largest shortcut between agents and other pivots until no more pivots provide a positive shortcut. Pivot pruning has a runtime complexity of $O(MP^3)$ where P is the number of pivots. Pivot pruning is run for every mTSP heuristic calculation prior to running the mTSP solver. Pivot pruning is not guaranteed to increase the heuristic value, and in some cases may even reduce the informativeness of the heuristic. An example is shown in Figure 4(d-f), where removing pivot p_3, which provides a shortcut from a to p_2, worsens the overall mTSP heuristic from 11 to 9. However, our experiments show that it provides a significant speedup.

3.3 Parallel Heuristic Calculation

Despite using lazy heuristic evaluation for A* expansions, the majority of the time overhead in MWRP-A* comes from the mTSP heuristic calculation. Parallelization algorithms for lazy heuristics have been explored for large-scale multiprocessing (Mukherjee, Aine, and Likhachev 2022), however these methods require constantly running asynchronous processes. Instead, we use Batch A* (Li et al. 2022; Agostinelli et al. 2019), which is often used to parallelize the

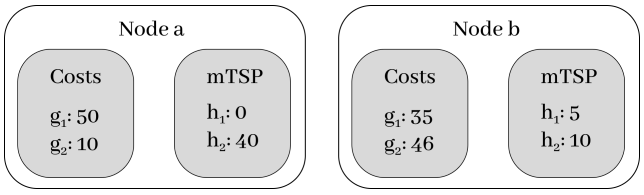


Figure 5: Two example search nodes, each with the agent costs (g_k) and estimates on the remaining search efforts (h_k).

calculation of neural network heuristics. During each expansion cycle of A*, we first check if the next node to be popped has had its mTSP heuristic computed. If not, we parallelize the computation by looping through the next N nodes that are going to be popped from *OPEN* and accumulating the nodes that have not yet had their mTSP heuristic computed into a *batch*. We then compute the mTSP heuristic for the batch in parallel and update each node's f-value in *OPEN*.

Since N is a constant, the runtime complexity of A* is not changed. Larger N values will parallelize more heuristic calculations, but may execute more unnecessary computations, since there is no guarantee that all of these nodes would have been expanded. We use $N = 100$ in our experiments.

As proven above, MWRP-CP^3 is guaranteed to return an optimal solution. Notably, while our focus is the makespan objective, MWRP-CP^3 makes no assumptions on the objective function and is thus optimal for graph-based WRP and MWRP with either the sum-of-costs or makespan objective.

4 Bounded Suboptimal Methods

Bounded Suboptimal Search (BSS) algorithms are a class of search methods that, given a weight $w \geq 1$, guarantee a solution with cost $C \leq C^* \cdot w$, where C^* is the cost of the optimal solution. We present two BSS algorithms that are used to scale MWRP-CP^3 to larger maps and agent counts.

4.1 Minimax Weighted A* (MxWA*)

A popular BSS variant of A* is weighted A* (WA*) (Pohl 1970). In WA*, the f-value is altered to $f_w(n) = g(n) + w \cdot h(n)$, where $g(n)$ is the cost to reach node n and $h(n)$ is an admissible heuristic. WA* has previously been applied to the graph-based WRP problem with WRP-A* (Yaffe, Skyler, and Felner 2021). To our knowledge, WA* has not been applied to makespan problems with joint-space search.

We propose *Minimax Weighted A** (MxWA*), a WA* variant specifically for the makespan objective. The goal is to prioritize agents with lower search effort while ensuring that the solution found is still bounded suboptimal. MxWA* first obtains a weighted f_k-value for each agent a_k. Then, the maximum of these f_k-values is used as the w-admissible f-value during the search. That is, MxWA* performs the A* search with the f-value $f_{MxW}(n) = \max_k\{g_k + w \cdot h_k\}$.

For example, consider nodes a and b shown in Figure 5. Node a has $g(a) = \max\{g_1, g_2\} = 50$ and admissible $f(a) = \max\{g_1 + h_1, g_2 + h_2\} = 50$. Similarly, node b has $g(b) = \max\{g_1, g_2\} = 45$ and admissible $f(b) =$

$\max\{g_1 + h_1, g_2 + h_2\} = 55$. In A*, node a is expanded first since it has a lower f-value.

However, using MxWA*, we generate $f_{MxW}(a) = \max\{50 + w \cdot 0, 10 + w \cdot 40\} = 10 + w \cdot 40$ and $f_{MxW}(b) = \max\{35 + w \cdot 5, 45 + w \cdot 10\} = 45 + w \cdot 10$. Thus, with $w \geq 1.2$, node b would be expanded first, which is preferred since it has lower h-values and thus has a lower remaining search effort. We now prove that MxWA* is w-suboptimal.

Theorem 3. *Given agent costs $g_k(n)$ and estimates of agent cost-to-go of $h_k(n)$, performing A* search with $f_{MxW}(n) = \max_i\{g_k() + w \cdot h_k(n)\}$ returns a solution with cost $C \leq w \cdot C^*$, where C^* is the cost of the optimal solution.*

Proof. We define the last node expanded during the search as node G with solution cost C. Let P^* be the optimal search path with cost C^*, and let n be the last node on P^* that is in *OPEN*. Since G was expanded before n, we know that $f_{MxW}(G) \leq f_{MxW}(n)$, and since $h_k(G) = 0$ for each agent a_k, $f_{MxW}(G) = g(G)$, so $g(G) \leq f_{MxW}(n) = \max_k\{g_k(n) + w \cdot h_k(n)\} \leq \max_k\{w \cdot g_k(n) + w \cdot h_k(n)\} = w \cdot f(n)$. This means $g(G) \leq w \cdot f(n)$. Since $f(n)$ is admissible and n is on P^*, we know $f(n) \leq C^*$, meaning $g(G) = C \leq w \cdot C^*$. □

In each node, g_k is the cumulative cost of agent a_k. For the Singleton heuristic, $h_{s,k}$ is the distance from agent a_k to the nearest watcher of s. Using MxWA*, $f_{s,MxW} = \min_k\{g_k + w \cdot h_{s,k}\}$ is a w-admissible f-value, so $f_{MxW} = \max_s\{f_{s,MxW}\}$. For the mTSP f-value, h_k is agent a_k's path length along G_{DLS}, so we compute f_{MxW} by minimizing $\max_k\{g_k + w \cdot h_k\}$ as the mTSP objective function.

In the single-agent case, MxWA* is identical to WA*. Additionally, following techniques from Anytime Weighted A* (Hansen and Zhou 2007), we extend MxWA* to Anytime MxWA* (AMxWA*). AMxWA* continues to search for better solutions even after an initial solution was found. If B is the cost of the best solution found by the search thus far, then AMxWA* prunes nodes n with $\max\{\frac{f_{MxW}(n)}{w}, g(n)\} \geq B$, since $\frac{f_{MxW}(n)}{w}$ is guaranteed to be admissible. This forces future solutions to improve on the current best solution.

4.2 Focal Search

The second BSS method utilizes *Focal Search* (FS), which uses both an admissible heuristic $h(n)$ and a second, not necessarily admissible, heuristic $h_{FOCAL}(n)$ that serves as a better estimate of search effort (Pearl and Kim 1982). FS maintains the *OPEN* list from A* as well as a separate *FOCAL* list. *FOCAL* contains the subset of nodes n in *OPEN* with $f(n) \leq w \cdot f_{min}$, where f_{min} is the current minimum f-value in *OPEN*. Each iteration, the node with the lowest $h_{FOCAL}(n)$ value from *FOCAL* is expanded.

Since we use lazy heuristic evaluation in *OPEN*, we need to recompute the heuristic of some nodes before inserting them into *FOCAL*. Each A* expansion cycle, all nodes n in *OPEN* that were inserted with the Singleton f-value $f(n) \leq w \cdot f_{min}$ are popped and reinserted into *OPEN* with the mTSP f-value. *FOCAL* contains the subset of nodes n in *OPEN* with an mTSP f-value of $f(n) \leq w \cdot f_{min}$. We propose two different h_{FOCAL} heuristics, both of which estimate the remaining search effort based on the paths produced by mTSP. The *Sum Of Remaining Costs* (SORC) heuristic calculates $\sum_k h_k$, and the *Max Of Remaining Costs* (MORC) heuristic calculates $\max_k h_k$.

We also utilize these heuristics with Anytime Focal Search (AFS) (Cohen et al. 2018). Defining B as the cost of the best solution found thus far, AFS limits the nodes in *FOCAL* to nodes with $f(n) < \min\{B, w \cdot f_{min}\}$ to force future solutions to improve on the current best solution.

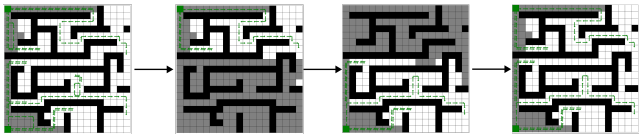


Figure 6: Visualization of the initial joint-space solution, decomposed agent sub-problems, and postprocessed solution.

5 Postprocessing Framework

Postprocessing is utilized in planning algorithms to quickly improve the solution quality of an existing suboptimal solution. We propose a postprocessing framework that decomposes the problem into several single-agent searches that are each run on a separate subset of the map. Naturally, to improve the makespan of an existing suboptimal solution, $\boldsymbol{\pi}$, we look to improve the path of the agent with the highest path cost while ensuring that the resulting set of paths is still a solution. Defining agent a_{max} as the agent with the largest path cost, we want to replace π_a, the path of a_{max}, with π'_a where (1) $c(\pi'_a) < c(\pi_a)$ and (2) the resulting set of paths still has line-of-sight to every cell in $\mathcal{U}$.

To ensure that the new set of paths has line-of-sight to every cell in $\mathcal{U}$, we need to ensure that π'_a sees all cells in $\mathcal{U}$ that no other agent sees. Thus, defining $\boldsymbol{\pi}^-$ as the paths of all agents other than a_{max}, we extract the *minimum responsibility* (r) of a_{max} as $r = \mathcal{U} \setminus \bigcup_{s \in \mathcal{P}(\boldsymbol{\pi}^-)} \mathcal{L}(s)$. Here, r represents the cells that only a_{max} sees in the original solution, and thus must see in any improved solution. We utilize MWRP-CP^3 to optimally solve a single-agent subproblem for agent a_{max} with $\mathcal{U} = r$ (Figure 6). Formally, we solve the problem tuple $(1, \mathcal{C}, r, \{\mathcal{S}_{a_{max}}\}, \mathcal{N}, \mathcal{L}, \mathcal{O})$.

Once a new path π'_a is computed, we update $\pi_a = \pi'_a$. We repeat this process until the agent with the largest cost among $\boldsymbol{\pi}$ has already had its path optimized in this decomposed manner (Figure 6). Since the agent responsibilities (r) may not be optimally partitioned, the postprocessing framework cannot guarantee a globally optimal solution.

6 Experimental Results

We conduct experiments with varying map types, map sizes, and agent counts to measure the performance of MWRP-CP^3, our BSS algorithms, and our postprocessing frame-

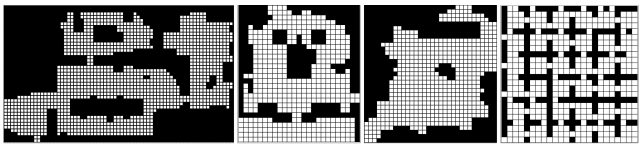


Figure 7: From left to right, Den101d, Lak105d, Den202d (Stern et al. 2019), and an example Room map.

Map Average initial size of $\|\mathcal{U}\|$		Random $\|\mathcal{U}\| = 175.3$	Maze $\|\mathcal{U}\| = 598.9$	Minecraft $\|\mathcal{U}\| = 1279.5$
CD	**% Reduction**	0.8 ± 0.6	69.9 ± 1.2	18.5 ± 2.3
	Runtime	2 ± 0	4 ± 0	45 ± 1
PD	**% Reduction**	$\mathbf{56.6 \pm 19.8}$	$\mathbf{95.3 \pm 1.1}$	$\mathbf{89.0 \pm 3.8}$
	Runtime	8 ± 0	28 ± 1	205 ± 3
CPD	**% Reduction**	$\mathbf{56.6 \pm 19.8}$	$\mathbf{95.3 \pm 1.1}$	$\mathbf{89.0 \pm 3.8}$
	Runtime	$\mathbf{6 \pm 1}$	$\mathbf{8 \pm 0}$	$\mathbf{160 \pm 2}$

Table 1: Comparison of CD, PD, and CPD on three distinct maps (shown in Figure 3). The % reduction in the size of $\mathcal{U}$, as well as the CPU runtime (ms), are measured. The initial $\mathcal{U}$ is C minus the initial LOS of each agent.

work. For all of our experiments, we use four-way movement as the neighbor function and BresenhamLOS as the LOS function (Figure 1). All algorithms, including the original MWRP-A* algorithm, are implemented in C++. The mTSP heuristic is implemented as an integer linear programming formulation (Bektas 2006) using CPLEX (cpl 2021). All experiments were run on a Legion 5i Gen 10 with 32 Intel Core Ultra 255Hx processors (4.50 GHz) and 32GB of RAM. We ran experiments on four different map styles (Figures 3 and 7): (1) Maze maps, (2) Room maps consisting of 3×3 rooms, (3) Random maps with 20% obstacles, and (4) Game-inspired maps (Figures 3 and 7). Room and random maps were custom-generated; the others were taken from the Moving AI MAPF Benchmark (Stern et al. 2019), previous work on MWRP (Livne et al. 2023), and previous work on multi-agent planning and task assignment (Chong, Li, and Sycara 2024). For all experiments, to simulate real-world scenarios where exploration requires entering an environment from the outside, we generate agent starting points by selecting random cells along the map border.

6.1 State Space Reduction

First, we compare the state space reduction and execution runtime of CD, PD, and CPD on three distinct map architectures (visualized in Figure 3). Averages were computed over 50 runs per agent count, from 1-5 agents, on each map. As shown in Table 1, PD and CPD were able to reduce $|\mathcal{U}|$ by more than 95% on some maps. CD and PD both pruned more cells in structured maps with long corridors, which are less prominent in *Random* maps and more prominent in *Maze* maps. CPD executed $1.25\times$ faster than PD.

6.2 Search Algorithm Runtime Comparison

To our knowledge, MWRP-A* is the only existing optimal algorithm for solving MWRP, and there are no bounded suboptimal algorithms. We compare runtime performance between MWRP-A*, MWRP-A* with CPD, MWRP-CP^3, MxWA*, FS with SORC, and FS with MORC. The runtime measurements are averaged over 20 instances, each with randomly generated agent starting locations. CPD runtime was included in the total runtime. As shown in Figure 8, MWRP-CP^3 is the fastest optimal solver, running $200\times$ faster than MWRP-A*, and solving complex problem instances that MWRP-A* failed to solve within a 200s time limit. The runtime improvement is significantly lower on *Random* maps, as CPD is less effective (Table 1). Figure 8 also demonstrates the scalability of our suboptimal solvers with respect to increasing map sizes and agent counts. For larger agent counts, MxWA* performs better than both FS algorithms, likely because the size of *FOCAL* increases with a larger joint-space, resulting in more mTSP heuristic calculations.

	32×32 Maze		**Den101d**	
	$M = 1$	$M = 3$	$M = 1$	$M = 3$
PP	3.21 ± 1.26	8.18 ± 6.35	1.38 ± 0.19	4.99 ± 2.15
PHC	2.03 ± 0.13	5.59 ± 2.00	1.76 ± 0.30	2.33 ± 0.83

Table 2: Ablation tests with M agents measuring the scalar factor increase in runtime after removing either PP or PHC.

6.3 Ablation Study

We also performed an ablation study to measure the individual improvement of the Pivot Pruning (PP) and Parallel Heuristic Calculation (PHC) techniques. Table 2 shows the slowdown caused by removing PP or PHC, each averaged over 20 different problem instances. Both methods provide a speedup in all test cases, and both provide a larger benefit on problems with more agents. PHC has low variances in its slowdown factor, indicating that its speedup is consistent regardless of the problem. PP, however, has high variances, meaning its speedup varies from problem to problem.

6.4 Comparison of Suboptimal Variants

As shown in Figure 9(a), we measured the performance of all three suboptimal methods at different weight values by running the algorithms on the same problem instance, a 32×32 Maze Map with 6 agents, at varying w values. The runtime and solution cost were averaged over 10 iterations. All three algorithms have increasing solution costs increasing and decreasing runtimes with larger w values. At lower w values, all three algorithms perform similarly in terms of runtime and solution cost. At higher w values, MxWA* is able to achieve faster runtimes than either FS.

We also empirically tested the anytime behavior of AMxWA*, AFS with SORC heuristic, and AFS with MORC heuristic, as shown in Figure 9(b). All three algorithms were run on the Minecraft-inspired map with 2 agents. The algorithms were run only once, since the improvement over time cannot be averaged over several runs. We can see that all three algorithms improve their costs significantly over the course of the runtime. AFS with the SORC heuristic takes longer than AMxWA* to generate an initial solution (for $w = 3$), but is able to quickly improve its solution much faster than AMxWA* and AFS with the MORC heuristic.

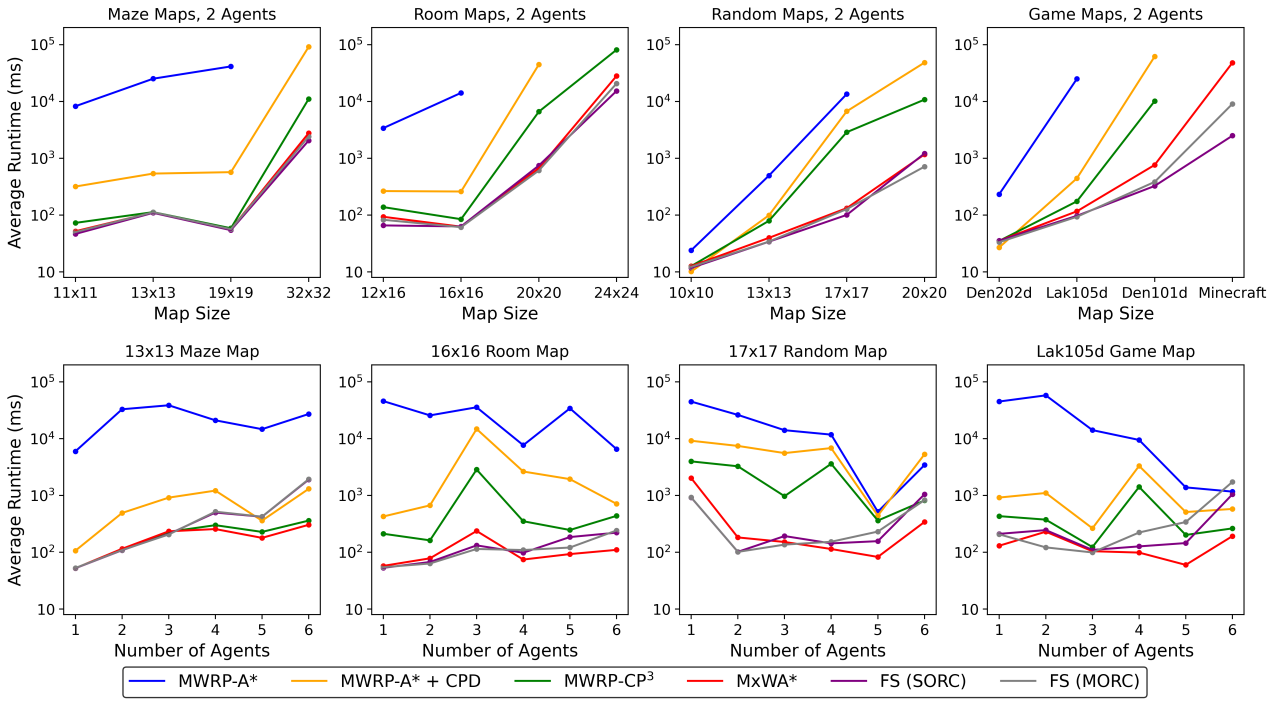
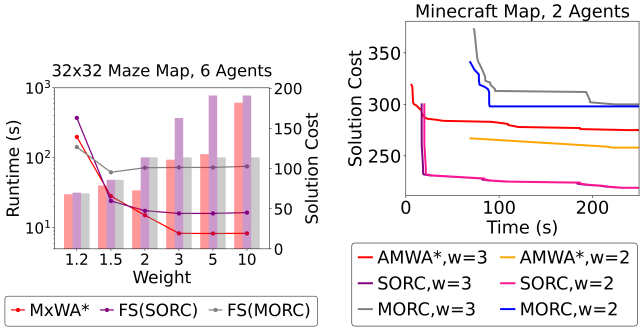


Figure 8: Runtime comparison of search algorithms with respect to map size (top) and agent counts (bottom). Suboptimal methods used $w = 2$. Algorithms were subject to a 200s time limit, and only those solving 80%+ of the instances are shown.



(a) (b)

Figure 9: Figure(a) shows runtime (lines) and solution cost (bars) at different w values. Figure(b) shows the cost improvement over time of the anytime variations.

These results demonstrate that all three algorithms are effective depending on the specific scenario and w value.

6.5 Impact of Postprocessing Framework

The postprocessing framework was tested on a 32×32 Maze with 3 agents. As shown in Table 3, for each algorithm, the postprocessing framework increased overall runtime by only 8% while improving the solution cost significantly, with MxWA* achieving a near-optimal postprocessed solution. In general, the postprocessing framework is best suited for MxWA* since its solution better distributes the paths throughout the map, which helps create even partitions.

	w	BSS Algorithm		Postprocessing	
		Runtime	Cost	Runtime	Cost
MWRP-CP^3	1	7,379	92	0	92
MxWA*	2	1,277	**124**	164	**93**
	5	1,154	129	146	109
FS (SORC)	2	1,121	162	168	125
	5	1,105	162	182	125
FS (MORC)	2	**1,074**	127	**156**	107
	5	1,091	127	166	107

Table 3: Impact of postprocessing architecture on BSS algorithms. The BSS runtime (ms), cost of the initial BSS solution, postprocessing runtime (ms), and improved solution cost are shown. We also show MWRP-CP^3 as a reference.

7 Conclusion

In this work, we introduced MWRP-CP^3, which optimally solves MWRP by utilizing cell and path dominance, pivot pruning, and parallel heuristic computation, and computes optimal paths 200× faster than existing baselines. We also introduced MxWA*, a variation of WA* for the makespan objective function, and SORC and MORC, two heuristics used in conjunction with Focal Search. Additionally, we presented anytime variations for our BSS algorithms, as well as a postprocessing framework, both of which significantly improve the quality of the suboptimal solution. Future work should (1) investigate faster heuristics (Ren, Rathinam, and Choset 2024), (2) experiment on non-grid graphs (Kavraki and Latombe 1994), and (3) expand the problem scope (i.e. considering inter-agent collisions or heterogeneous agents).

Acknowledgments

The research at Carnegie Mellon University was partially supported by the National Science Foundation under grants #2328671 and #2441629. Ariel Felner was supported by Israel Science Foundation (ISF) Grant #909/23 and by a grant from the Israeli Ministry of Science and Technology (MOST).

References

2021. *IBM ILOG CPLEX Optimization Studio*. International Business Machines Corporation.

Agostinelli, F.; McAleer, S.; Shmakov, A.; and Baldi, P. 2019. Solving the Rubik's cube with Deep Reinforcement Learning and Search. *Nature Machine Intelligence*, 1(8): 356–363.

Bektas, T. 2006. The Multiple Traveling Salesman Problem: An Overview of Formulations and Solution Procedures. *Omega*, 34(3): 209–219.

Bezas, K.; Tsoumanis, G.; Angelis, C. T.; and Oikonomou, K. 2022. Coverage Path Planning and Point-of-interest Detection using Autonomous Drone Swarms. *Sensors*, 22(19): 7551.

Bresenham, J. E. 1965. Algorithm for Computer Control of a Digital Plotter. *IBM Systems Journal*, 4(1): 25–30.

Cho, S. W.; Park, H. J.; Lee, H.; Shim, D. H.; and Kim, S.-Y. 2021. Coverage Path Planning for Multiple Unmanned Aerial Vehicles in Maritime Search and Rescue Operations. *Computers Industrial Engineering*, 161: 107612.

Chong, Y. Q.; Li, J.; and Sycara, K. 2024. Optimal Task Assignment and Path Planning using Conflict-Based Search with Precedence and Temporal Constraints. *arXiv preprint arXiv:2402.08772*.

Cohen, L.; Greco, M.; Ma, H.; Hernandez, C.; Felner, A.; Kumar, T. K. S.; and Koenig, S. 2018. Anytime Focal Search with Applications. In *Proceedings of the International Joint Conference on Artificial Intelligence*, 1434–1441.

Gavish, B. 1976. Note—A Note on "The Formulation of the M-Salesman Traveling Salesman Problem". *Management Science*, 22(6): 704–705.

Hansen, E. A.; and Zhou, R. 2007. Anytime Heuristic Search. *Journal of Artificial Intelligence Research*, 28: 267–297.

Huang, W. H. 2001. Optimal Line-sweep-based Decompositions for Coverage Algorithms. In *Proceedings of the International Conference on Robotics and Automation*, volume 1, 27–32.

Jimenez, P. A.; Shirinzadeh, B.; Nicholson, A.; and Alici, G. 2007. Optimal Area Covering using Genetic Algorithms. In *Proceedings of the International Conference on Advanced Intelligent Mechatronics*, 1–5.

Kavraki, L.; and Latombe, J.-C. 1994. Randomized preprocessing of configuration for fast path planning. In *Proceedings of the International Conference on Robotics and Automation*, 2138–2145.

Li, T.; Chen, R.; Mavrin, B.; Sturtevant, N. R.; Nadav, D.; and Felner, A. 2022. Optimal Search with Neural Networks: Challenges and Approaches. In *Proceedings of the International Symposium on Combinatorial Search*, 109–117.

Livne, Y.; Atzmon, D.; Skyler, S.; Boyarski, E.; Shapiro, A.; and Felner, A. 2023. Optimally Solving the Multiple Watchman Route Problem with Heuristic Search. In *Proceedings of the International Conference on Autonomous Agents and Multiagent Systems*, 905–913.

Mero, L. 1984. A Heuristic Search Algorithm with Modifiable Estimate. *Artificial Intelligence*, 23(1): 13–27.

Mitchell, J. S.; and Wynters, E. 1991. Watchman Routes for Multiple Guards. In *Proceedings of the Canadian Conference on Computational Geometry*, 126–129.

Mukherjee, S.; Aine, S.; and Likhachev, M. 2022. MPLP: Massively Parallelized Lazy Planning. *IEEE Robotics and Automation Letters*, 7(3): 6067–6074.

Nilsson, B. 1995. *Guarding Art Galleries - Methods for Mobile Guards*. Ph.D. thesis, Lund University.

Packer, E. 2008. Computing multiple watchman routes. In *International Workshop on Experimental and Efficient Algorithms*, 114–128.

pang Chin, W.; and Ntafos, S. 1988. Optimum Watchman Routes. *Information Processing Letters*, 28(1): 39–44.

Pearl, J.; and Kim, J. H. 1982. Studies in Semi-Admissible Heuristics. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PAMI-4(4): 392–399.

Pohl, I. 1970. Heuristic Search Viewed as Path Finding in a Graph. *Artificial Intelligence*, 1(3-4): 193–204.

Ren, Z.; Rathinam, S.; and Choset, H. 2024. A Bounded Sub-optimal Approach for Multi-agent Combinatorial Path Finding. *IEEE Transactions on Automation Science and Engineering*, 22: 7590–7605.

Seiref, S.; Jaffey, T.; Lopatin, M.; and Felner, A. 2020. Solving the Watchman Route Problem on a Grid with Heuristic Search. In *Proceedings of the International Conference on Automated Planning and Scheduling*, 249–257.

Steinbrink, M.; Koch, P.; Jung, B.; and May, S. 2021. Rapidly-exploring Random Graph Next-best View Exploration for Ground Vehicles. In *2021 European Conference on Mobile Robots (ECMR)*, 1–7.

Stern, R.; Sturtevant, N. R.; Felner, A.; Koenig, S.; Ma, H.; Walker, T. T.; Li, J.; Atzmon, D.; Cohen, L.; Kumar, T. K. S.; Boyarski, E.; and Bartak, R. 2019. Multi-Agent Pathfinding: Definitions, Variants, and Benchmarks. In *Proceedings of the International Symposium on Combinatorial Search*, 151–158.

Yaffe, T.; Skyler, S.; and Felner, A. 2021. Suboptimally Solving the Watchman Route Problem on a Grid with Heuristic Search. In *Proceedings of the International Symposium on Combinatorial Search*, 106–114.

Yamauchi, B. 1997. A Frontier-based Approach for Autonomous Exploration. In *Proceedings of the International Symposium on Computational Intelligence in Robotics and Automation*, 146–151.