# A Complete and Bounded-Suboptimal Algorithm for a Moving Target Traveling Salesman Problem with Obstacles in 3D*

Anoop Bhat[1] and Geordan Gutow[1] and Bhaskar Vundurthy[1] and
Zhongqiang Ren[2] and Sivakumar Rathinam[3] and Howie Choset[1]

***Abstract*— The moving target traveling salesman problem with obstacles (MT-TSP-O) seeks an obstacle-free trajectory for an agent that intercepts a given set of moving targets, each within specified time windows, and returns to the agent's starting position. Each target moves with a constant velocity within its time windows, and the agent has a speed limit no smaller than any target's speed. We present FMC*-TSP, the first complete and bounded-suboptimal algorithm for the MT-TSP-O, and results for an agent whose configuration space is $\mathbb{R}^3$. Our algorithm interleaves a high-level search and a low-level search, where the high-level search solves a generalized traveling salesman problem with time windows (GTSP-TW) to find a sequence of targets and corresponding time windows for the agent to visit. Given such a sequence, the low-level search then finds an associated agent trajectory. To solve the low-level planning problem, we develop a new algorithm called FMC*, which finds a shortest path on a graph of convex sets (GCS) via implicit graph search and pruning techniques specialized for problems with moving targets. We test FMC*-TSP on 280 problem instances with up to 40 targets and demonstrate its smaller median runtime than a baseline based on prior work.**

## I. INTRODUCTION

Visiting moving targets in environments with obstacles is necessary in applications ranging from underway replenishment of naval ships [1] to delivering spare parts to spacecraft on-orbit [2]. The shortest path problem of visiting multiple targets by an agent is often modeled as a traveling salesman problem (TSP) in the literature [3], [4]. Given the cost of travel between any pair of targets, the TSP aims to find a sequence of targets to visit such that each target is visited exactly once and the sum of the travel costs for the agent is minimized. The moving target TSP (MT-TSP) is a generalization of the TSP where the targets are mobile, and may only be visited within specific time windows. The MT-TSP seeks a sequence of targets as well as a trajectory through space for an agent that intercepts each target. When the agent must avoid static obstacles, we have the moving target TSP with obstacles (MT-TSP-O) (refer to Fig. 1).

As with other motion planning problems, two desirable properties in an algorithm for the MT-TSP-O are completeness and optimality. No existing MT-TSP-O algorithm has both properties. Even finding a feasible solution to the MT-TSP-O is NP-complete [5] due to the presence of time windows, making completeness difficult to guarantee. Furthermore, since the MT-TSP-O generalizes the TSP, solving the MT-TSP-O optimally is NP-hard. Our prior work [6] developed a complete algorithm for the MT-TSP-O in $\mathbb{R}^2$ using a generalization of a visibility graph. The completeness of [6] is limited to the plane, where a shortest path in a visibility graph is also a shortest path in continuous space. This property fails in three dimensions[1]. Note that with MT-TSP-O, path length affects completeness because of timing constraints: if an agent takes an excessively long path from its starting point to a destination point on a target's trajectory, the agent may not reach its destination point by the time the target gets there, even if doing so is possible.

In this paper, we propose the first complete and bounded-suboptimal algorithm for the MT-TSP-O in three-dimensions. Our algorithm interleaves a high-level search for a sequence of targets and associated time windows, and a low-level search for a trajectory intercepting a given target/time window sequence. The low-level search uses a graph of convex sets (GCS) [7], rather than the visibility-based graph in [6],

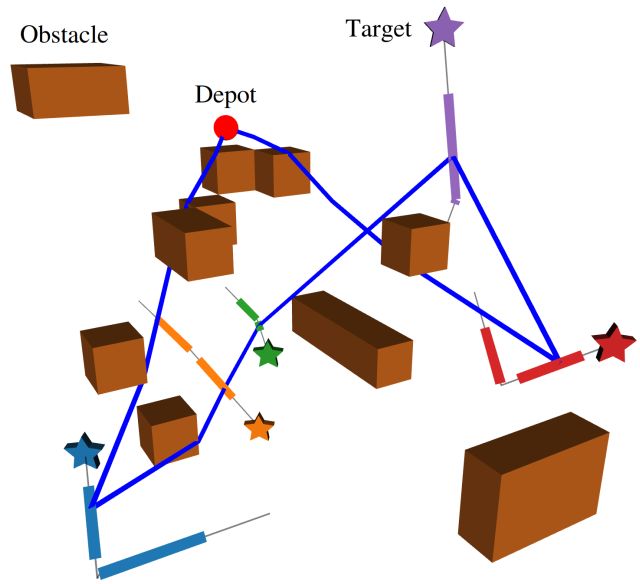


Fig. 1. Targets depicted as stars move along piecewise-linear trajectories. Portions of a target's trajectory within time windows are highlighted in bold colors. Agent trajectory shown in blue begins and ends at depot, intercepting each target within one of its time windows while avoiding obstacles.

[1]Robotics Institute at Carnegie Mellon University, 5000 Forbes Ave., Pittsburgh, PA 15213, USA. Emails: {agbhat, ggutow, pvundurt, choset}@andrew.cmu.edu. This research was supported by an appointment to the Intelligence Community Postdoctoral Research Fellowship Program.
[2]UM-SJTU Joint Institute and Department of Automation at Shanghai Jiao Tong University, Shanghai, China. Email: zhongqiang.ren@sjtu.edu.cn
[3]Department of Mechanical Engineering and Department of Computer Science and Engineering at Texas A&M University, College Station, TX 77843. Email: srathinam@tamu.edu

[1]As in figure 1, the agent's path from the blue target to the green target wraps around an edge of an obstacle rather than a vertex. A visibility graph only considers paths that move between obstacle vertices.

because the visibility graph approach is limited to the plane. In particular, the low-level search extends existing techniques for implicit graph search on a GCS [8], [9], using focal search [10] to enforce bounded-suboptimality and additional limits on arrival times to intermediate targets to prune suboptimal solutions. Therefore, one of our contributions is a new GCS-based algorithm for intercepting a given sequence of moving targets in minimum-time, which we call FMC* (Focal search for Moving targets on a graph of Convex sets). We call our overall algorithm FMC*-TSP. We test FMC*-TSP against a baseline based on prior work that samples the trajectories of targets into points. The baseline is not complete or bounded-suboptimal, and often needs a large number of samples to find a solution. As we increase the number of targets, FMC*-TSP often finds solutions for the MT-TSP-O more quickly compared to the baseline while satisfying a specified suboptimality bound.

## II. RELATED WORK

Several algorithms exist for the MT-TSP without obstacles, ranging from complete and optimal methods [11]–[14] to incomplete and suboptimal heuristic methods [2], [15]–[22]. The complete and optimal methods assume trajectories of targets are linear or piecewise-linear; our work makes the same piecewise-linear assumption. In the presence of obstacles, there are two related works. [23] represents trajectories of targets using sample points and requires a straight line agent trajectory between each sample point: this method is neither complete nor optimal. Our prior work [6] develops a complete (but not an optimal) algorithm for the MT-TSP-O, for planar instances with piecewise-linear target trajectories.

A complete and optimal method of planning trajectories in non-planar environments is to plan a shortest path on a graph of convex sets (GCS) [7], [24]. [7] finds the shortest path on a GCS via a mixed integer convex program (MICP), which can be solved exactly to guarantee completeness and optimality, or solved via the relaxation-then-rounding method from [24] that sacrifices guarantees to improve runtime. IxG* [8] and GCS* [9] replace the MICP in [7] with implicit graph search, allowing operations on larger graphs. The low-level planner in our work also implicitly searches a GCS, but exploits the structure of minimum-time problems with moving targets and time windows to accelerate the search.

## III. PROBLEM SETUP

Let $\mathcal{Q}_{free}$ be the obstacle-free configuration space in $\mathcal{Q} = \mathbb{R}^3$. For some final time $t_f$, let the agent's trajectory be $\tau_a : [0, t_f] \to \mathcal{Q}$. For all $t \in [0, t_f]$, we require $\tau_a(t) \in \mathcal{Q}_{free}$, $\tau_a(0) = p_d$ (the position of the depot), and that $\tau_a$ is *speed admissible*, *i.e.* it never exceeds the agent's speed limit $v_{max}$.

Denote the set of moving targets as $\mathcal{I} = \{1, 2, \dots, |\mathcal{I}|\}$. Each target $i \in \mathcal{I}$ is associated with a set of time windows, $\mathcal{W}_i = \{[\underline{t}_i^1, \overline{t}_i^1], [\underline{t}_i^2, \overline{t}_i^2], \dots, [\underline{t}_i^{|\mathcal{W}_i|}, \overline{t}_i^{|\mathcal{W}_i|}]\}$. $\underline{t}_i^j$ and $\overline{t}_i^j$ denote the start and end time of target $i$'s $j$th time window respectively. Let the trajectory of target $i$ be $\tau_i : \mathbb{R} \to \mathcal{Q}$. We assume within each of target $i$'s time windows, $\tau_i$ lies in $\mathcal{Q}_{free}$ and has speed no larger than $v_{max}$. We also assume $\tau_i$ has constant velocity within each time window in $\mathcal{W}_i$, though the velocity may differ from one window to another.

We define a *target-window* $u_i^j$ as the pairing of target $i$ with its $j$th time window, i.e. $u_i^j = (i, [\underline{t}_i^j, \overline{t}_i^j])$. We also define a target-window associated with the depot, $u_d = u_0^1 = (0, [0, \infty))$. Here, we treat the depot as a fictitious target 0 with $\tau_0(t) = p_d$ for all $t$, and with $\mathcal{W}_0 = \{[0, \infty)\}$. Let $\mathcal{T}_i = \{i\} \times \mathcal{W}_i$ be the set of target-windows for target $i$. For each target-window $u_i^j$, we let $\texttt{Tar}(u_i^j) = i$, $\underline{t}(u_i^j) = \underline{t}_i^j$, $\overline{t}(u_i^j) = \overline{t}_i^j$, and $\texttt{Traj}(u_i^j) = \tau_i$. An agent trajectory $\tau_a$ *intercepts* target-window $u$ at time $t$ if $u_a(t) = \texttt{Traj}(u)(t)$.

A *tour* is a sequence of target-windows containing exactly one target-window per target, beginning and ending with $u_d$. A *partial tour* is a sequence of target-windows beginning with $u_d$ and containing at most one target-window per target, not necessarily ending with $u_d$. For a partial tour $\Gamma$, $|\Gamma|$ is the length, $\Gamma[i]$ is the $i$th element[2], and $\Gamma[-1]$ is the final element. Additionally, the length-$i$ *prefix* of $\Gamma$ is $\Gamma[: i] = (\Gamma[1], \Gamma[2], \dots, \Gamma[i])$. A partial tour $\Gamma$ *visits* target $i$ if for some $u \in \mathcal{T}_i$, $u \in \Gamma$. An agent trajectory $\tau_a$ *executes* a partial tour $\Gamma$ if $\tau_a$ intercepts the target-windows in $\Gamma$ in order. The MT-TSP-O seeks a tour $\Gamma$, an agent trajectory $\tau_a$, and a final time $t_f$ such that $\tau_a$ executes $\Gamma$, and $t_f$ is minimized. In this work, we assume that we are given a suboptimality factor $w$, such that if the optimal value of $t_f$ is $t_f^*$, then our returned solution $(\Gamma, \tau_a, t_f)$ satisfies $t_f \le w t_f^*$.

## IV. FMC*-TSP ALGORITHM

An overview of FMC*-TSP is shown in Alg. 1. FMC*-TSP searches for a tour on a target-window graph (Section IV-A), where the nodes are target-windows, and an edge connects target-window $u$ to $v$ if the agent can intercept $u$, then $v$, in the absence of obstacles. The search for a tour is posed as a generalized traveling salesman problem with time windows (GTSP-TW) (Section IV-B). For every tour generated by the GTSP-TW solver, FMC*-TSP performs a low-level search on a GCS for a trajectory executing the tour (Sections IV-C to IV-D). The low-level search updates an incumbent solution $(\Gamma^{inc}, \tau_a^{inc}, t_f^{inc})$ (Line 2) with the lowest-cost trajectory found so far. Whenever the low-level search finds a trajectory executing a tour $\Gamma$, it adds $\Gamma$ to the *forbidden set* $\mathcal{H}_{forbid}$ (Line 3), forbidding the GTSP-TW solver from producing $\Gamma$ again. On the other hand, if the low-level search fails to find a trajectory for a tour $\Gamma$, it adds the shortest prefix of $\Gamma$ that cannot be executed to $\mathcal{H}_{forbid}$.

FMC*-TSP ensures bounded-suboptimality by terminating only when the incumbent satisfies $t_f^{inc} \le wLB$, where $LB$ is a lower bound on the optimal cost $t_f^*$. $LB$ is the minimum of two values, $LB_H$ and $LB_L$. $LB_H$ is a value maintained by the GTSP-TW solver, lower bounding the cost of any tour the solver can currently produce. $LB_L$ is a value updated by the low-level search. In particular, whenever the low-level search produces a trajectory for a tour $\Gamma$, it also produces a lower bound on the cost of executing $\Gamma$, and $LB_L$ is continually updated to be the lowest of these lower bounds.

[2]We use 1-based indexing, i.e. the first element of $\Gamma$ is $\Gamma[1]$

Finally, for pruning purposes, FMC*-TSP maintains a dictionary called $\mathfrak{D}$. The keys for $\mathfrak{D}$ are tuples $(\mathcal{S}, u)$, where $\mathcal{S} \subseteq \mathcal{I}$, and $u$ is a target-window. The value for a key $(\mathcal{S}, u)$ is the cost of the best trajectory found so far intercepting the targets in $\mathcal{S}$, in any order, and terminating by intercepting $u$. We define the function $\texttt{Key}(\Gamma)$, which converts a partial tour $\Gamma$ into a key $(\mathcal{S}, \Gamma[-1])$, where $S$ contains the targets visited by $\Gamma$. Whenever we generate a trajectory for a partial tour $\Gamma$, we constrain the trajectory to execute $\Gamma$ within time $\mathfrak{D}[\texttt{Key}(\Gamma)]$ and prune $\Gamma$ if this is infeasible. If we attempt to access $\mathfrak{D}[(S, u)]$ for some key $(S, u)$ not contained in $\mathfrak{D}$, we assume $\mathfrak{D}[(S, u)]$ evaluates to $\bar{t}(u)$.

**Algorithm 1:** FMC*-TSP

```
G_tw = ConstructTargetWindowGraph ();
(Γ^inc, τ_a^inc, t_f^inc) = (NULL, NULL, ∞);
H_forbid = {};
LB_L = ∞;
𝔇 = dict();
SolveGTSP-TW (G_tw, LowLevelSearch);
if t_f = ∞ then return INFEASIBLE;
return (Γ^inc, τ_a^inc, t_f^inc);
```

### A. Target-Window Graph

We define a graph $\mathcal{G}_{tw} = (\mathcal{V}_{tw}, \mathcal{E}_{tw})$, called a *target-window graph*. $\mathcal{V}_{tw}$ is the set of nodes, containing all target-windows. $\mathcal{E}_{tw}$ is the set of edges. To define the edges, we need the following quantities:

**Definition 1.** *Given target-windows $u$ and $v$, the* obstacle-unaware latest feasible departure time $\mathbf{l}_{uv}$ *is the maximum $t \in [\underline{t}(u), \bar{t}(u)]$ such that a speed-admissible trajectory exists intercepting $u$ at some $t_0 \in [\underline{t}(u), \bar{t}(u)]$, then $v$ at $t$, in the absence of obstacles. If no such trajectory exists, $\mathbf{l}_{uv} = -\infty$.*

**Definition 2.** *Given target-windows $u$ and $v$ and departure time $t_0$ from $u$, the* obstacle-unaware earliest feasible arrival time $\mathbf{e}_{uv}(t_0)$ *is the minimum $t \in [\underline{t}(v), \bar{t}(v)]$ such that a speed-admissible trajectory exists intercepting $u$ at time $t_0$ then $v$ at time $t$, in the absence of obstacles. If no such trajectory exists, $\mathbf{e}_{uv}(t_0) = \infty$. Additionally, when we write $\mathbf{e}_{uv}$ without any argument, we imply the argument $t_0 = \underline{t}(u)$.*

**Definition 3.** *Given target-windows $u$ and $v$, the* obstacle-unaware shortest feasible travel time $\mathbf{s}_{uv}$ *equals* $\min_{t_0 \in [\underline{t}(u), \bar{t}(u)]}(\mathbf{e}_{uv}(t_0) - t_0)$. *If $\mathbf{e}_{uv} = \infty$, $\mathbf{s}_{uv} = \infty$.*

$\mathbf{l}_{uv}$, $\mathbf{e}_{uv}$, and $\mathbf{s}_{uv}$ have closed-form expressions [25]. For all pairs of target-windows $(u, v)$ with $\texttt{Tar}(u) \neq \texttt{Tar}(v)$, we first compute $\mathbf{l}_{uv}$; if $\mathbf{l}_{uv} \neq -\infty$, we compute $\mathbf{e}_{uv}$ and $\mathbf{s}_{uv}$, then store $\mathbf{l}_{uv}, \mathbf{e}_{uv}$ and $\mathbf{s}_{uv}$ in an edge $(u, v) \in \mathcal{E}_{tw}$. If $\mathbf{l}_{uv} = -\infty$, then $(u, v) \notin \mathcal{E}_{tw}$. For edge $e = (u, v) \in \mathcal{E}_{tw}$, let $\mathbf{l}_e = \mathbf{l}_{uv}$, $\mathbf{e}_e = \mathbf{e}_{uv}$ and $\mathbf{s}_e = \mathbf{s}_{uv}$.

### B. GTSP-TW

We solve a GTSP-TW on the target-window graph for a tour, using the mixed integer linear program (MILP) formulation from [26], modified to incorporate $\mathbf{l}_e, \mathbf{e}_e$, and $\mathbf{s}_e$ values. The decision variables consist of a binary variable $x_e \in \{0, 1\}$ for each edge $e \in \mathcal{E}_{tw}$, indicating whether $e$ is traveled in the tour, as well as an arrival time $t^i$ for each $i \in \mathcal{I} \cup \{0\}$. For a target-window $v$, $\delta^-(v)$ is the set of edges entering $v$, and $\delta^+(v)$ is the set of edges leaving $v$. Let $\delta(i, j) = \bigcup_{u \in \mathcal{T}_i} \bigcup_{v \in \mathcal{T}_j} (\delta^+(u) \cap \delta^-(v))$. The MILP is given in Problem 1.

$$\min_{\substack{\{x_e\}_{e \in \mathcal{E}_{tw}}, \\ \{t^i\}_{i \in \mathcal{I} \cup \{0\}}}} \quad t^0 \tag{1a}$$

$$\text{s.t.} \quad \sum_{v \in \mathcal{T}_i} \sum_{e \in \delta^-(v)} x_e = 1 \; \forall i \in \mathcal{I} \cup \{0\} \tag{1b}$$

$$\sum_{e \in \delta^-(v)} x_e = \sum_{e \in \delta^+(v)} x_e \; \forall v \in \mathcal{V}_{tw} \tag{1c}$$

$$\sum_{v \in \mathcal{T}_i} \sum_{e \in \delta^-(v)} \mathbf{e}_e x_e \leq t^i \leq \sum_{u \in \mathcal{T}_i} \sum_{e \in \delta^+(u)} \mathbf{l}_e x_e \; \forall i \in \mathcal{I} \tag{1d}$$

$$[i \neq 0] t^i - t^j + \sum_{e \in \delta(i,j)} \mathbf{s}_e x_e \leq \sum_{u \in \mathcal{T}_i} \sum_{e \in \delta^+(u)} \mathbf{l}_e x_e - \sum_{v \in \mathcal{T}_j} \sum_{e \in \delta^-(v)} \mathbf{e}_e x_e - \sum_{e \in \delta(i,j)} (\mathbf{l}_e - \mathbf{e}_e) x_e \quad \forall i \in \mathcal{I} \cup \{0\}, j \in \mathcal{I} \tag{1e}$$

$$t^i + \sum_{e \in \delta^-(u_d)} \mathbf{s}_e x_e \leq t^0 \; \forall i \in \mathcal{I} \tag{1f}$$

$$\sum_{e \in \text{edges}(\Gamma)} x_e \leq |\Gamma| - 2 \; \forall \Gamma \in \mathcal{H}_{forbid} \tag{1g}$$

(1a) minimizes the arrival time at the depot. (1b) requires the tour to visit all targets and the depot. (1c) requires that if the tour arrives at a target-window $v$, the tour also departs from $v$. (1d) expresses that the arrival time at target $i$ is no earlier than $\mathbf{e}_e$ for the edge $e$ chosen to arrive at $i$, and that the arrival time is no later than $\mathbf{l}_e$ for the edge $e$ chosen to depart from $i$. (1e) ensures that if a tour contains $j$ immediately after $i$, the arrival time at $j$ is no earlier than the arrival time at $i$ plus $\mathbf{s}_e$ for the edge chosen to travel from $i$ to $j$. In (1e), $[i \neq 0]$ equals 1 if $i \neq 0$ and equals 0 otherwise, enforcing departure from the depot at $t = 0$. (1f) enforces a similar constraint to (1e) for traveling to the depot. (1g) requires that any partial tour $\Gamma$ in $\mathcal{H}_{forbid}$ is not a prefix of a returned solution[3].

### C. Low-Level Search

When solving the MILP 1, a standard solver will produce several tours. For each tour $\Gamma$, the low-level search (LLS) attempts to generate a trajectory executing $\Gamma$, as well as a lower bound $\underline{g}(\Gamma)$ on the cost to execute $\Gamma$. LLS does so by computing a trajectory and lower bound for each prefix of $\Gamma$ that it has not seen before, in increasing order of prefix length, using FMC* (Section IV-D). If trajectory computation fails for a prefix $\Gamma[:i]$ (i.e. FMC* returns $\underline{g}(\Gamma[:i]) = \infty$), we add $\Gamma[:i]$ to $\mathcal{H}_{forbid}$ and return to the high-level search.

[3]For a partial tour $\Gamma = (u^1, u^2, \ldots, u^{|\Gamma|})$, we define edges$(\Gamma) = \{(u^1, u^2), (u^2, u^3), \ldots, (u^{|\Gamma|-1}, u^{|\Gamma|})\}$

If we successfully generate a trajectory for $\Gamma$, we update the incumbent with $\Gamma$, and we update $LB_L$ using $\underline{g}(\Gamma)$.

**Algorithm 2:** `LowLevelSearch` $(\Gamma)$

1 **for** $i$ in $(2, 3, \ldots, |\Gamma|)$ **do**
2     **if** `SawBefore` $(\Gamma[: i])$ **then** continue;
3     $(\underline{g}(\Gamma[: i]), \tau_a, t_f)$ = `FMC*` $(\Gamma[: i])$;
4     $\mathcal{H}_{forbid} \leftarrow \mathcal{H}_{forbid} \cup \{\Gamma[: i]\}$;
5     **if** $\underline{g}(\Gamma[: i]) = \infty$ **then** return ;
6 $(\Gamma^{inc}, \tau_a^{inc}, t_f^{inc}) \leftarrow (\Gamma, \tau_a, t_f)$;
7 $LB_L \leftarrow \min(LB_L, \underline{g}(\Gamma))$;

### D. FMC*

The low-level search invokes FMC* with a partial tour $\Omega$ (the same as $\Gamma[: i]$ in Alg. 2) and seeks a trajectory executing $\Omega$, as well as a lower bound on its execution cost $\underline{g}(\Omega)$. Before invoking FMC* for the first time, we decompose the obstacle-free configuration space $\mathcal{Q}_{free}$ into convex regions $\{\mathcal{A}^1, \mathcal{A}^2, \ldots, \mathcal{A}^{n_{reg}}\}$. Since the obstacle maps in our experiments are cubic grids, we choose to decompose $\mathcal{Q}_{free}$ into axis-aligned bounding boxes, but other decomposition methods, e.g. [27], can be used as well. We then define a graph of convex sets (GCS), denoted as $G_{cs} = (\mathcal{V}_{cs}, \mathcal{E}_{cs})$, where $\mathcal{V}_{cs}$ is the set of nodes and $\mathcal{E}_{cs}$ is the set of edges. Each node in $\mathcal{V}_{cs}$ is a convex subset of $\mathcal{Q} \times \mathbb{R}$, i.e. a set of configurations and times. For each convex region $\mathcal{A}$ decomposing $\mathcal{Q}_{free}$, we have a *region-node* $\mathcal{X}_{\mathcal{A}} = \mathcal{A} \times [-\infty, \infty]$ in $\mathcal{V}_{cs}$. For each target-window $u$, we have a *window-node* $\mathcal{X}_u \in \mathcal{V}_{cs}$ containing the positions and times along $\texttt{Traj}(u)$ within time interval $[\underline{t}(u), \bar{t}(u)]$: this set of positions and times is a line segment in space-time, and thereby a convex set [11]. We say an agent trajectory *intercepts* a window-node if it intercepts the associated target-window. An edge connects $\mathcal{X}$ to $\mathcal{X}'$ if $\mathcal{X} \cap \mathcal{X}' \neq \emptyset$. We call a sequence of GCS nodes $P$ a *path*. Similar to the notation in Section III, $|P|$ is the length of $P$, $P[j]$ is the $j$th element of $P$, and $P[-1]$ is the final element of $P$. We say path $P$ is a *solution path* for $\Omega$ if $(\mathcal{X}_{\Omega[1]}, \mathcal{X}_{\Omega[2]}, \ldots \mathcal{X}_{\Omega[-1]})$ is a subsequence of $P$. FMC* interleaves the construction of a solution path and the optimization of an associated trajectory.

FMC* finds a path and trajectory using focal search. The search maintains two priority queues, OPEN and FOCAL, each containing paths. Each path $P \in$ OPEN has an $f$-value $f(P)$, lower bounding the cost of a solution path beginning with $P$. We construct $f(P)$ as the sum of a cost-to-come $g(P)$ and a cost-to-go $h(P)$. OPEN prioritizes paths with smaller $f$-values. Let $f_{min}(\text{OPEN}) = \min_{v \in \text{OPEN}} f(v)$, and let $f_{min}(\text{OPEN}) = \infty$ if OPEN is empty. $f_{min}(\text{OPEN})$ is a lower bound on the cost of executing $\Omega$. When FMC* returns, along with returning a trajectory, it returns $f_{min}(\text{OPEN})$, which becomes $\underline{g}(\Gamma[: i])$ in Alg. 2. In addition to its $f$-value, a path $P$ also stores an obstacle-free, speed-admissible agent trajectory $\texttt{LTraj}(P)$ and a time $\texttt{LTime}(P)$, such that $\texttt{LTraj}(P)$ intercepts all window-nodes in $P$ in order, and $\texttt{LTime}(P)$ is the interception time of the last window-node in $P$ ("L" stands for last).

The priority queue FOCAL stores all $P \in$ OPEN with $f(v) \leq w f_{min}(\text{OPEN})$. Each $P \in$ FOCAL also has a priority vector $\vec{f}_{pr}(P) = [\text{unv}(P), g + wh(P)]^T$, where $\text{unv}(P)$ is the number of target-windows in $\Omega$ without an associated GCS node in $P$, i.e. unvisited by $P$. $g + wh(P)$ is the priority function from weighted A* [28]. FOCAL prioritizes nodes with lexicographically smaller priority vectors, i.e. it prefers paths with fewer unvisited target-windows, and for two paths with the same number of unvisited target-windows, FOCAL prioritizes them in the same way as weighted A*.

When invoking FMC* in Alg 2, note that if $|\Omega| > 2$, we must have already computed a trajectory for $\Omega[: |\Omega| - 1]$. We can reuse the OPEN list from this previous search to speed up the current one (Line 5), and this reuse is described in Section IV-D.1. If $|\Omega| = 2$, we cannot reuse a previous OPEN list, so we initialize OPEN as empty, then add a path to OPEN containing only the depot (Lines 1-3).

Each search iteration pops a path $P$ from FOCAL, then obtains $\texttt{NextWindowIndex}(\Omega, P)$, which is the smallest index $n$ such that $\mathcal{X}_{\Omega[n]} \notin P$. Next, for any $j \in \{n, n+1, \ldots |\Omega|\}$, we check if $\texttt{Key}(\Omega[: j])$ is in $\mathfrak{D}$, and if so, we check if $\mathfrak{D}[\texttt{Key}(\Omega[: j])]$ is equal to the start of the time window for $\Omega[j]$ (Line 12). If so, there is no use finding a trajectory for $\Omega[: j]$, and we prune $P$[4]. If we have not pruned $P$, we obtain the *successor nodes* of $P$. $\mathcal{X}' \in \mathcal{V}_{cs}$ is a successor node of $P$ if the following conditions hold:

1) $(P[-1], \mathcal{X}') \in \mathcal{E}_{cs}$
2) $\mathcal{X}'$ does not occur in $P$ after the final window-node in $P$, ensuring that descendants of $P$ will not visit a node more than once between visits to two window-nodes
3) If $\mathcal{X}'$ is a window-node, $\mathcal{X}' = \mathcal{X}_{\Omega[n]}$

From each successor node $\mathcal{X}'$ of $P$, we create a *successor path* $P'$ by appending $\mathcal{X}'$ to $P$.

We now describe how to compute $f(P')$ and $\vec{f}_{pr}(P')$. The procedure is illustrated in Fig. 2. We first obtain the index of the final window-node in $P$: call this index $j$. We then construct an auxiliary path, $P_{aux} = (P[j], P[j+1], \ldots, P[-1], \mathcal{Q} \times \mathbb{R})$. Next, we optimize a speed-admissible trajectory, parameterized as a sequence of contiguous linear trajectory segments $(\tau^1, \tau^2, \ldots, \tau^{|P_{aux}|})$ and a sequence of segment end times $(t^1, t^2, \ldots, t^{|P_{aux}|})$, such that $t^{|P_{aux}|}$ is minimized, segment $k$ lies entirely in set $P_{aux}[k]$, $\tau^1(\texttt{LTime}(P)) = \texttt{LTraj}(P)(\texttt{LTime}(P))$, and $\tau^{|P_{aux}|}(t^{|P_{aux}|}) = \texttt{Traj}(\Omega[n])(t^{|P_{aux}|})$. This optimization is the same as [8], eqn. 2, with the additional initial and terminal constraints. If the optimal final time $t^{|P_{aux}|}$ (denoted as $t$ on Line 15) is larger than $\mathfrak{D}[\texttt{Key}(\Omega[: n])]$, we prune $P'$. Otherwise, we note that if we concatenate $\texttt{LTraj}(P)$ with segments $k = 1$ to $k = |P_{aux}| - 1$, we obtain an obstacle-free trajectory (denoted as $\tau'$ on Line 15). The cost of $\tau'$ is $t^{|P_{aux}|-1}$, and we let $g(P') = t^{|P_{aux}|-1}$. If $P'[-1] = \mathcal{X}_{\Omega[n]}$, $\tau'$ is a new trajectory $\tau_{\Omega[:n]}$ executing $\Omega[: n]$, and we update $\mathfrak{D}[\texttt{Key}(\Omega[: n])]$ (Line 18) to prune future trajectories worse than $\tau_{\Omega[:n]}$. Next, we note that $\tau^{|P_{aux}|}$ travels in a straight

[4] In practice, to account for numerical tolerances, we use the condition $\mathfrak{D}[\texttt{Key}(\Omega[: j])] \leq \underline{t}(\Omega[j]) + \epsilon$, with $\epsilon = 10^{-8}$.

line to position $\texttt{Traj}(\Omega[n])(t^{|P_{aux}|})$, ignoring obstacles, terminating at time $t^{|P_{aux}|}$, denoted as $t$ on Line 15. Lines 23-26 extend $\tau^{|P_{aux}|}$ to intercept all remaining target-windows $\Omega[j]$ in $\Omega$ by successively computing the obstacle-unaware earliest feasible arrival time from one target-window to the next. If the arrival time at any $\Omega[j]$ is larger than $\mathfrak{D}[\texttt{Key}(\Omega[:j])]$, we prune $P'$. Otherwise, $f(P')$ is the arrival time at $\Omega[-1]$, and $h(P') = f(P') - g(P')$, letting us compute $\vec{f}_{pr}$.

**Algorithm 3:** $\texttt{FMC}\star$ ($\Omega$)

1 **if** $|\Omega| = 2$ **then**
2 OPEN = [];
3 Push $P = (\mathcal{X}_{u_d})$ onto OPEN with $f(P) = 0$;
4 **else**
5 OPEN = $\texttt{ReuseOpen}$ ($\Omega[:|\Omega|-1]$, $\Omega$);
6 $\tau_\Omega$ = NULL;
7 **while** OPEN is not empty and ($\tau_\Omega$ is NULL or $\mathfrak{D}[\texttt{Key}(\Omega)] > wf_{min}$(OPEN)) **do**
8 FOCAL = $\{P \in \text{OPEN} : f(P) \leq f_{min}(\text{OPEN})\}$;
9 $P$ = FOCAL.pop();
10 OPEN.remove($P$);
11 $n$ = $\texttt{NextWindowIndex}$ ($\Omega, P$);
12 **if** $\exists j \in \{n, n+1, \dots |\Omega|\}$ s.t. $\texttt{Key}(\Omega[:j])$ in $\mathfrak{D}$ and $\mathfrak{D}[\texttt{Key}(\Omega[:j])] = \underline{t}(\Gamma[j])$ **then** continue;
13 **for** $\mathcal{X}'$ in $\texttt{SuccessorNodes}$ ($P$) **do**
14 $P' = \texttt{Append}(P, \mathcal{X}')$;
15 $t, g(P'), \tau'$ = $\texttt{OptimizeTrajectory}$ ($P'$);
16 **if** $t > \mathfrak{D}[\texttt{Key}(\Omega[:n])]$ **then** continue ;
17 **if** $\mathcal{X}' = \mathcal{X}_{\Omega[n]}$ **then**
18 $\tau_{\Omega[:n]} = \tau'$, $\mathfrak{D}[\texttt{Key}(\Omega[:n])] = g(P')$;
19 $\texttt{LTraj}(P') = \tau'$, $\texttt{LTime}(P') = g(P')$;
20 **else**
21 $\texttt{LTraj}(P') = \texttt{LTraj}(P)$;
22 $\texttt{LTime}(P') = \texttt{LTime}(P)$;
23 **for** $j$ in $(n+1, n+2, \dots, |\Omega|)$ **do**
24 $t \leftarrow \mathbf{e}_{\Omega[j-1]\Omega[j]}(t)$;
25 **if** $t > \mathfrak{D}[\texttt{Key}(\Omega[:j])]$ **then** break ;
26 **if** $t > \mathfrak{D}[\texttt{Key}(\Omega[:j])]$ **then** continue ;
27 Push $P'$ onto OPEN with $f(P') = t$;
28 $h(P') = f(P') - g(P')$;
29 $\vec{f}_{pr}(P') = [\text{unv}(P'), g(P') + wh(P')]^T$;
30 return $f_{min}$(OPEN), $\tau_\Omega$, $\mathfrak{D}[\texttt{Key}(\Omega)]$;

*1) Reusing OPEN Between FMC* Calls:* When invoking FMC* with $|\Omega| > 2$, we set OPEN equal to the OPEN list at the end of the FMC* search for $\Omega[:|\Omega|-1]$, with updated $f$-values and $\vec{f}_{pr}$ vectors. For each $P \in$ OPEN, before updating $f(P)$ and $\vec{f}_{pr}(P)$, we execute Lines 11-12 and prune $P$ if the condition on Line 12 holds. If we did not prune $P$, we perform the update $f(P) \leftarrow \mathbf{e}_{\Omega[-2]\Omega[-1]}(f(P))$. We then update $\vec{f}_{pr}(P)$ by executing lines 28-29 with $P' = P$.

## V. THEORETICAL ANALYSIS

In this section, we sketch proofs for the bounded-suboptimality and completeness of FMC*-TSP.

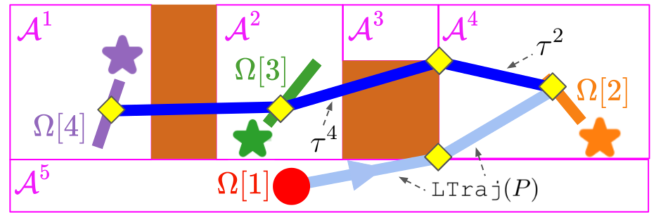


Fig. 2. Computing $f(P')$ for $P' = (\mathcal{X}_{\Omega[1]}, \mathcal{X}_{\mathcal{A}^5}, \mathcal{X}_{\mathcal{A}^4}, \mathcal{X}_{\Omega[2]}, \mathcal{X}_{\mathcal{A}^4}.\mathcal{X}_{\mathcal{A}^3})$. $P_{aux} = (\mathcal{X}_{\Omega[2]}, \mathcal{X}_{\mathcal{A}^4}, \mathcal{X}_{\mathcal{A}^3}, \mathcal{Q} \times \mathbb{R})$. Regions decomposing $\mathcal{Q}_{free}$ are shown as pink boxes. Trajectory segment endpoints are shown as yellow diamonds. $\tau^1$ is a zero-length segment between $\texttt{LTraj}(P)$ and $\tau^2$, and $\tau^3$ is a zero-length segment between $\tau^2$ and $\tau^4$. Concatenating $\texttt{LTraj}(P)$ with $\tau^1, \tau^2$, and $\tau^3$ gives an obstacle-free trajectory $\tau'$ with cost $g(P')$. $\tau^4$ and its extension ignore obstacles. Extension of $\tau^4$ terminates on purple target at time $f(P')$.

**Theorem 1.** *For a feasible problem instance, Alg. 1 returns a solution with cost no more than $w$ times the optimal cost.*

Let $\Gamma^*$ be a tour whose minimum execution cost is equal to the optimal cost of a problem instance, $t_f^*$. At any time, either $\Gamma^*$ corresponds to a feasible solution to Problem 1, meaning $LB_H$ lower bounds the execution cost of $\Gamma^*$, or the low-level search found a trajectory for $\Gamma^*$, meaning that $LB_L$ lower bounds the execution cost of $\Gamma^*$. This means $LB = \min(LB_H, LB_L) \leq t_f^*$. By terminating only when the incumbent satisfies $t_f^{inc} \leq wLB$, we ensure that $t_f^{inc} \leq wt_f^*$.

**Theorem 2.** *For an infeasible problem instance, Alg. 1 will report that the instance is infeasible in finite time.*

Finite-time termination follows from the finite number of tours Problem 1 can generate and the finite number of paths that Alg. 3 can explore. Since no feasible solution exists, the incumbent cost $t_f^{inc}$ remains at its default value $\infty$, and Alg. 1 must return infeasible on Line 7.

## VI. NUMERICAL RESULTS

We ran experiments on an Intel i9-9820X 3.3GHz CPU with 128 GB RAM. Experiment 1 (Section VI-A) varied the number of targets and the sum of their time window lengths, Experiment 2 (Section VI-B) varied the agent's speed limit, and Experiment 3 (Section VI-C) varied FMC*-TSP's suboptimality factor. All problem instances had 2 time windows per target. We generated 280 total instances by extending the instance generation method from [6] to 3D. We compared FMC*-TSP to a baseline based on related work [12], [23], [25], which samples trajectories of targets into points and solves a generalized traveling salesman problem (GTSP) to find a sequence of points to visit. The baseline solves its GTSP via the integer program [29] without subtour elimination constraints. Subtours are only possible if two samples are identical, which we ensure never occurs. Costs between sample points are initially an obstacle-unaware cost, and whenever the GTSP solver generates a sequence of points, we evaluate the obstacle-aware costs between consecutive pairs of points in parallel using a variant of IxG* [8]. We limited each method's computation time to 60 s.

When we compared with the baseline (Experiments 1 and 2), we set a suboptimality factor $w = 1.1$ for FMC*-TSP. The baseline is not bounded-suboptimal for the MT-TSP-O,

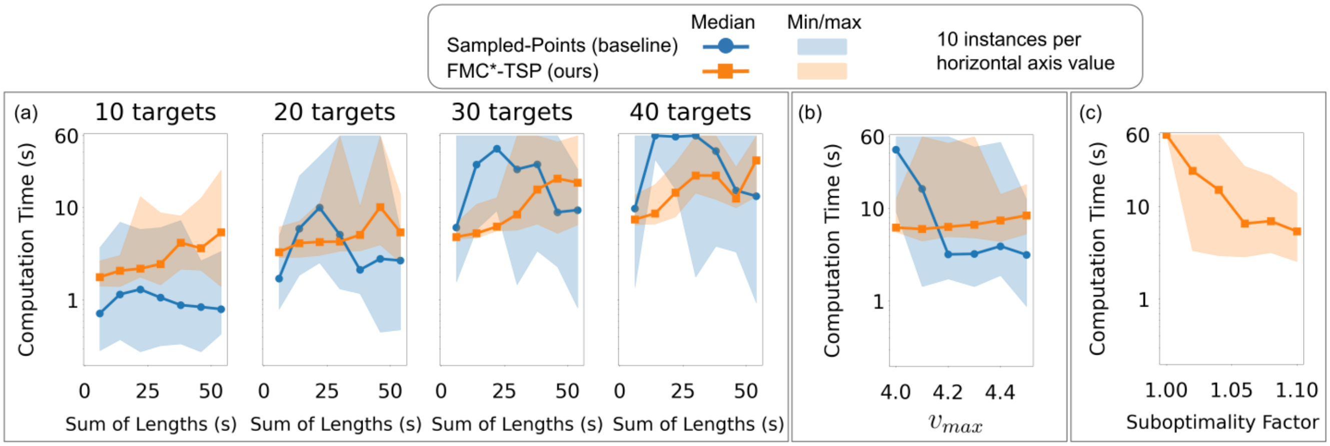


Fig. 3. All vertical axes are on a log-scale. (a) Varying the number of targets and sum of time window lengths per target. (b) Varying the agent's speed limit. (c) Varying FMC*-TSP's suboptimality factor.

but it is bounded-suboptimal for the sampled approximation of the MT-TSP-O that it solves, and we set a suboptimality factor of 1.1 to make a roughly fair comparison with FMC*-TSP. The baseline is also not complete, but if it uses more sample points, it is more likely to find a solution. Therefore, we initialized the baseline with 5 points per target, and whenever the baseline failed to find a solution, we added 5 more points per target and tried again.

### A. Experiment 1: Varying Sum of Time Window Lengths

We varied the number of targets from 10 to 40 and the sum of time window lengths per target from 6 s to 54 s. The results are in Fig. 3 (a). As we saw in [6], there is a range for the sum of window lengths where the baseline takes more median computation time than FMC*-TSP to find a solution, and this range widens as we increase the number of targets. For these ranges, the baseline needs an excessive number of sample points to find a feasible solution, since the fraction of a target's time windows that is part of a feasible solution is small. FMC*-TSP's computation time increases as we increase the sum of lengths, similar to previous results on TSP variants with time windows [11], [30]. Runtime increased in both major components of FMC*-TSP: the GTSP-TW mixed integer program and the low-level FMC* search. GTSP-TW runtime increases with time window length because when we have larger time windows, it is feasible more often to travel from one target-window to another, leading to more edges in the target-window graph and thereby more binary variables in Problem 1. FMC* computation time increases because FMC*-TSP calls FMC* more times: a larger number of edges in the target-window graph causes the GTSP-TW to produce more tours and seek trajectories for them from FMC*. When we increased the sum of lengths from 6 to 54, the median ratio of GTSP-TW time to FMC* time went from 0.076 to 0.19, i.e. while GTSP-TW time increased, FMC* remained the bottleneck.

In Table I, we show statistics for the percent difference in cost between FMC*-TSP and the baseline in cases where the both methods found a solution. Since we set FMC*-TSP's suboptimality factor to 1.1, its cost is never more than 10% larger than the baseline's, and at best, its cost is 21% lower.

TABLE I

COMPARISON OF THE SOLUTION COST $t_{\text{FMC*-TSP}}$ FROM FMC*-TSP AGAINST THE COST $t_{\text{SP}}$ FROM THE SAMPLED-POINTS METHOD.

| | Median | Min | Max |
|---|---|---|---|
| $\frac{t_{\text{FMC*-TSP}} - t_{\text{SP}}}{t_{\text{SP}}} * 100\%$ | **-0.035%** | **-21%** | **8.2%** |

### B. Experiment 2: Varying Agent's Speed Limit

We varied the agent's speed limit $v_{max}$ from 4.0 to 4.5, fixing the number of targets to 30 and the sum of lengths per target to 22 s. The results are in Fig. 3 (b). As we increase $v_{max}$, the baseline's computation time decreases. This is because if the agent can move faster, the subintervals of targets' time windows that are part of a feasible solution grow larger, and it is more likely that a sample point lies within one of these subintervals. FMC*-TSP's median computation time slightly increases as we increase $v_{max}$, due to an increase in the size of the time window graph, similar to Experiment 1.

### C. Experiment 3: Varying Suboptimality Factor

We varied the suboptimality factor of FMC*-TSP from $w = 1.0$ to $w = 1.1$, fixing the number of targets to 20 and the sum of lengths to 54 s. The results are shown in Fig. 3 (c). As expected, as the allowable suboptimality increases, FMC*-TSP computes solutions more quickly.

## VII. CONCLUSION

In this paper, we developed FMC*-TSP, a complete and bounded-suboptimal algorithm for the MT-TSP-O in 3D that leverages graphs of convex sets. We demonstrated a range of time window lengths for the targets where FMC*-TSP finds solutions more quickly than a baseline that samples trajectories of targets into points. A natural extension of this work would be to incorporate multiple agents.

## ACKNOWLEDGMENT

This material is partially based on work supported by the National Science Foundation (NSF) under Grant No. 2120219 and 2120529. Any opinions, findings, conclusions, or recommendations expressed in this material are those of the author(s) and do not necessarily reflect views of the NSF.

# References


[1] G. G. Brown, W. C. DeGrange, W. L. Price, and A. A. Rowe, "Scheduling combat logistics force replenishments at sea for the us navy," *Naval Research Logistics (NRL)*, vol. 64, no. 8, pp. 677–693, 2017.

[2] J.-M. Bourjolly, O. Gurtuna, and A. Lyngvi, "On-orbit servicing: a time-dependent, moving-target traveling salesman problem," *International Transactions in Operational Research*, vol. 13, no. 5, pp. 461–481, 2006.

[3] W. J. Cook, D. L. Applegate, R. E. Bixby, and V. Chvatal, *The traveling salesman problem: a computational study*. Princeton university press, 2011.

[4] G. Gutin and A. P. Punnen, *The traveling salesman problem and its variations*. Springer Science & Business Media, 2006, vol. 12.

[5] M. W. Savelsbergh, "Local search in routing problems with time windows," *Annals of Operations research*, vol. 4, pp. 285–305, 1985.

[6] A. Bhat, G. Gutow, B. Vundurthy, Z. Ren, S. Rathinam, and H. Choset, "A complete algorithm for a moving target traveling salesman problem with obstacles," in *International Workshop on the Algorithmic Foundations of Robotics*. Springer, 2024.

[7] T. Marcucci, J. Umenberger, P. Parrilo, and R. Tedrake, "Shortest paths in graphs of convex sets," *SIAM Journal on Optimization*, vol. 34, no. 1, pp. 507–532, 2024.

[8] R. Natarajan, C. Liu, H. Choset, and M. Likhachev, "Implicit graph search for planning on graphs of convex sets," *Robotics: Science and Systems (RSS)*, 2024.

[9] S. Y. C. Chia, R. H. Jiang, B. P. Graesdal, L. P. Kaelbling, and R. Tedrake, "Gcs*: Forward heuristic search on implicit graphs of convex sets," 2024. [Online]. Available: https://arxiv.org/abs/2407.08848

[10] J. Pearl and J. H. Kim, "Studies in semi-admissible heuristics," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. PAMI-4, no. 4, pp. 392–399, 1982.

[11] A. G. Philip, Z. Ren, S. Rathinam, and H. Choset, "A mixed-integer conic program for the moving-target traveling salesman problem based on a graph of convex sets," *arXiv preprint arXiv:2403.04917*, 2024.

[12] A. Stieber and A. Fügenschuh, "Dealing with time in the multiple traveling salespersons problem with moving targets," *Central European Journal of Operations Research*, vol. 30, no. 3, pp. 991–1017, 2022.

[13] A. Stieber, "The multiple traveling salesperson problem with moving targets," Ph.D. dissertation, BTU Cottbus-Senftenberg, 2022.

[14] A. G. Philip, Z. Ren, S. Rathinam, and H. Choset, "A mixed-integer conic program for the multi-agent moving-target traveling salesman problem," *arXiv preprint arXiv:2501.06130*, 2025.

[15] N. S. Choubey, "Moving target travelling salesman problem using genetic algorithm," *International Journal of Computer Applications*, vol. 70, no. 2, 2013.

[16] C. Groba, A. Sartal, and X. H. Vázquez, "Solving the dynamic traveling salesman problem using a genetic algorithm with trajectory prediction: An application to fish aggregating devices," *Computers & Operations Research*, vol. 56, pp. 22–32, 2015.

[17] U. Ucar and S. K. Işleyen, "A meta-heuristic solution approach for the destruction of moving targets through air operations." *International Journal of Industrial Engineering*, vol. 26, no. 6, 2019.

[18] Q. Jiang, R. Sarker, and H. Abbass, "Tracking moving targets and the non-stationary traveling salesman problem," *Complexity International*, vol. 11, no. 2005, pp. 171–179, 2005.

[19] R. S. de Moraes and E. P. de Freitas, "Experimental analysis of heuristic solutions for the moving target traveling salesman problem applied to a moving targets monitoring system," *Expert Systems with Applications*, vol. 136, pp. 392–409, 2019.

[20] B. Englot, T. Sahai, and I. Cohen, "Efficient tracking and pursuit of moving targets by heuristic solution of the traveling salesman problem," in *52nd IEEE Conference on Decision and Control*, 2013, pp. 3433–3438.

[21] D. Marlow, P. Kilby, and G. Mercer, "The travelling salesman problem in maritime surveillance–techniques, algorithms and analysis," in *Proceedings of the international congress on modelling and simulation*. Citeseer, 2007, pp. 684–690.

[22] Y. Wang and N. Wang, "Moving-target travelling salesman problem for a helicopter patrolling suspicious boats in antipiracy escort operations," *Expert Systems with Applications*, vol. 213, p. 118986, 2023.

[23] B. Li, B. R. Page, J. Hoffman, B. Moridian, and N. Mahmoudian, "Rendezvous planning for multiple auvs with mobile charging stations in dynamic currents," *IEEE Robotics and Automation Letters*, vol. 4, no. 2, pp. 1653–1660, 2019.

[24] T. Marcucci, M. Petersen, D. von Wrangel, and R. Tedrake, "Motion planning around obstacles with convex optimization," *Science robotics*, vol. 8, no. 84, p. eadf7843, 2023.

[25] A. G. Philip, Z. Ren, S. Rathinam, and H. Choset, "C*: A new bounding approach for the moving-target traveling salesman problem," *arXiv preprint arXiv:2312.05499*, 2023.

[26] Y. Yuan, D. Cattaruzza, M. Ogier, and F. Semet, "A branch-and-cut algorithm for the generalized traveling salesman problem with time windows," *European Journal of Operational Research*, vol. 286, no. 3, pp. 849–866, 2020.

[27] R. Deits and R. Tedrake, "Computing large convex regions of obstacle-free space through semidefinite programming," in *Algorithmic Foundations of Robotics XI: Selected Contributions of the Eleventh International Workshop on the Algorithmic Foundations of Robotics*. Springer, 2015, pp. 109–124.

[28] I. Pohl, "Heuristic search viewed as path finding in a graph," *Artificial intelligence*, vol. 1, no. 3-4, pp. 193–204, 1970.

[29] G. Laporte, H. Mercure, and Y. Nobert, "Generalized travelling salesman problem through n sets of nodes: the asymmetrical case," *Discrete Applied Mathematics*, vol. 18, no. 2, pp. 185–197, 1987.

[30] Y. Dumas, J. Desrosiers, E. Gelinas, and M. M. Solomon, "An optimal algorithm for the traveling salesman problem with time windows," *Operations research*, vol. 43, no. 2, pp. 367–371, 1995.