TMSTC*: A Path Planning Algorithm for Minimizing Turns in Multi-robot Coverage

Junjie Lu[1], Bi Zeng[1], Jingtao Tang[2], Tin Lun Lam[2] and Junbin Wen[1]

***Abstract*—Coverage path planning is a major application for mobile robots, which requires robots to move along a planned path to cover the entire map. For large-scale tasks, multirobot systems offer significant advantages. In this paper, we propose Turn-minimizing Multirobot Spanning Tree Coverage Star (TMSTC*), an improved multirobot coverage path planning (mCPP) algorithm based on MSTC*. Our algorithm partitions the map into minimum bricks as tree branches, transforming the problem into finding the maximum independent set of a bipartite graph. We then use a greedy strategy to connect bricks and form a tree, aiming to minimize the number of turns of the corresponding circumnavigating coverage path. Our experimental results show that our approach enables multiple robots to make fewer turns, resulting in faster completion of coverage tasks compared to other popular algorithms.**

***Index Terms*—Path Planning for Multiple Mobile Robots or Agents; Multi-Robot Systems; Service Robotics**

I. Introduction

COVERAGE path planning (CPP) involves mobile robots that traverse a target environment within their physical contact or sensor sensing range. The planning performance is typically measured by three metrics: coverage time, repeat coverage area size, and coverage rate [1]. CPP finds numerous applications in various fields, including military, agriculture, industry, commerce, disaster relief, and urban life. Some practical examples include automated cleaning [2], crop harvesting [3], adversarial patrolling [4], and robotic demining [5]. Due to its broad range of applications, coverage path planning research has received extensive attention.

Multi-robot systems are often employed in coverage tasks to achieve greater efficiency, especially as overlay size increases and multi-robot technology advances. Collaborative robots can perform tasks in parallel, resulting in faster completion times and improved fault tolerance through redundancy and task decomposition. They can also incorporate sensor information to compensate for localization uncertainty, improving overall efficiency. However, using teams of robots for coverage presents several challenges, including communication, localization, collaboration, and conflict resolution. To overcome these challenges, many approaches extend single-robot algorithms to multiple robots by decomposing and distributing the workload [6].

In coverage tasks, path length is typically used as the only cost metric, assuming constant robot speed. However, in reality, deceleration during turns increases completion time as the number of turns increases. For algorithms based on spanning tree coverage, the tree structure can significantly impact the number of turns. However, minimizing turns while reducing path length is an NP-Hard problem, even in 2D grid graphs [7]. To address this, we propose a new method called TMSTC* (Turn-minimizing Multirobot Spanning Tree Coverage Star) as an extension to MSTC* [8] to reduce turns and minimize completion time through averaging robot path cost with turns. We seek spanning trees with ideal shapes to enable multirobot coverage with fewer turns. Our contributions are summarized below.

1) We propose an effective algorithm to partition the map using a minimum number of bricks as spanning tree's branches. Here, a brick is defined as a rectangle with twice the width of a robot's covering tool.
2) We propose a greedy algorithm that first connects bricks to form a spanning tree and then applies the MSTC* strategy to find the optimal equilibrium division on a topological loop, resulting in an average weight of each robot's coverage path. The weights of the paths also take into account the turn cost.
3) We present extensive simulation in various environments. The results show that our method outperforms other popular STC-based algorithms and the state-of-the-art method TMC from [9] in terms of minimizing the number of turns and the coverage time.
4) We provide an open-source ROS package of our algorithm at https://github.com/CourierLo/TMSTC-Star.

II. Related Work

Coverage algorithms for robots can be classified into two categories: the exact decomposition method and the grid-based method. The former's representative algorithm is the Boustrophedon Cell Decomposition (BCD) [10], which decomposes the map into non-overlapping cells and makes the robot sweep back and forth along the coverage lines within these cells. BCD has variants that reduce the number of

Manuscript received: April, 24, 2023; Accepted June, 27, 2023.

This paper was recommended for publication by Editor M. Ani Hsieh upon evaluation of the Associate Editor and Reviewers' comments. This work was supported in part by the National Joint Fund Key Project (NSFC- Guangdong Joint Fund) under Grant U21A20478, the National Science Foundation of China under Grant 62172111, and in part by the Major Science and Technology Projects of Zhongshan under Grant 191018182628219. *(Corresponding author: Bi Zeng)*

[1]Junjie Lu, Bi Zeng and Junbin Wen are with the School of Computer Science and Technology, Guangdong University of Technology, Guangzhou 510006, China (e-mail: keithlo224@gmail.com; zb9215@gdut.edu.cn; xiosi12@163.com).

[2]Jingtao Tang and Tin Lun Lam are with the School of Science and Engineering, The Chinese University of Hong Kong, Shenzhen 518172, China (e-mail: todd.j.tang@gmail.com; tllam@cuhk.edu.cn).

Digital Object Identifier (DOI): see top of this page.

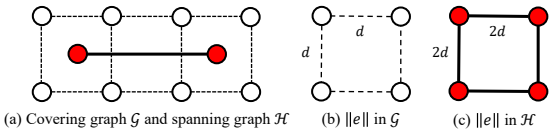
turns by modifying the sweep direction of each cell, such as measuring the geometric properties of each cell to minimize the sum of altitudes of the original BCD decomposition [11], [12], or minimizing a cost function that measures the robot's turning [13]. Although these variants of BCD can reduce the number of turns of the robot by using multiple coverage directions, cell decomposition is still time-intensive for complex and unstructured real environments. Vandermeulen et al. proposed a checkerboard decomposition of the environment to obtain rectangular cells and used an asymptotically partitioning heuristic to adjust the orientation of the rectangles to reduce the number of turns [9]. However, this iterative heuristic does not guarantee optimality and is also not suitable for non-rectilinear environments. Ramesh et al. developed the OARP coverage planning method, which aims to minimize axis-parallel coverage lines for a single robot coverage task by using a linear program (LP) [14], but the LP can become computationally intensive when dealing with large-scale problems. In this paper, we aim to overcome these limitations by dividing the environment using a minimum brick tiling that reduces the total number of turns without requiring complex cell decomposition.

Another type of grid-based approach decomposes the environment into a set of uniform grid cells, and the robot must find a path that covers all accessible grid cells. The method proposed in this paper belongs to this category. The shape of the grid cell is usually square, but there are also methods that use hexagons [15]. A fundamental method that has attracted much attention is the spanning tree coverage (STC) for a single robot proposed by Gabriely [16]. STC uses mega cells and a spanning tree to cover them all, with four smaller cells inside each mega cell accessible via the tree traversal. There are many variants of STC for multiple robots, with some methods changing the shape of the spanning tree to average the workload of each robot [17], some allocating sub-trees with similar weights to different robots to complete cooperative coverage [18], and some assigning equally-sized areas to each robot and then executing single-robot STC [19], [20]. Tang et al. proposed MSTC*, which uses a novel strategy to allocate coverage paths for multiple robots based on STC, and performs best among these STC variants for multiple robots [8]. However, these variants did not consider the impact of turning on coverage time and path partition. Our method will reduce the number of turns for robots overall and incorporate the cost of turning into the allocation strategy of MSTC*.

III. Problem Definition & Preliminaries

A. mCPP Problem Definition

The objective of mCPP is to use multiple (k) robots to cover all locations within a known terrain. For our purposes, we focus on robots that operate on flat surfaces, such as cleaning robots. Assuming the robot's covering tool has a width of d, we discretize the terrain into large square grid cells with a width of $2d$, which contains four small squares of width d. This allows us to create two graphs: the coverage graph $\mathcal{G}$ and the spanning graph $\mathcal{H}$. The nodes in these graphs are coverage nodes π and spanning nodes ρ, respectively. Coverage nodes correspond to grid cells with side length d, while spanning nodes correspond to larger grid cells with side length $2d$. The structure of these two graphs and the corresponding edge weights $\|e\|$ are depicted in Fig. 1.

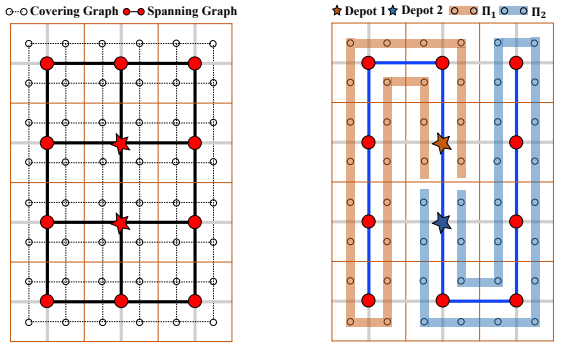


(a) Covering graph $\mathcal{G}$ and spanning graph $\mathcal{H}$ (b) $\|e\|$ in $\mathcal{G}$ (c) $\|e\|$ in $\mathcal{H}$

Fig. 1: The covering graph $\mathcal{G}$ is depicted using white nodes and dotted edges, while the spanning graph $\mathcal{H}$ is represented by red nodes and solid edges. Each spanning node is generated by aggregating four adjacent covering nodes. The edge length of $\mathcal{G}$ is set to d, while that of $\mathcal{H}$ is set to $2d$.

Given $\mathcal{G}$, $\mathcal{H}$, and k robots, we let the set of nodes through which robot r_i passes be $\{\pi_i^j\}$, and these nodes are from the path Π_i (see Fig. 2). The cost of the path is denoted by $\mathcal{W}_{\Pi_i}$. Ideally, each robot has the same workload and completes its respective coverage tasks at the same time. Therefore, we aim to find a set of coverage paths $\{\Pi_1, \Pi_2, \cdots, \Pi_k\}$ for k robots while minimizing the maximum weight of $\mathcal{W}_{\Pi_i}$. We want to ensure that each robot completes its task within a similar time, and the overall time taken to complete the coverage is minimized, that is

$$\arg\min_{\{\Pi_i\}} \left(\max_{1 \le i \le k} (\mathcal{W}_{\Pi_i}) \right) \tag{1}$$



(a) Covering Graph and Spanning Graph (b) Multi-robot Coverage Path Planning

Fig. 2: Multi-robot coverage path planning using our method TM-STC*. (a) Covering graph $\mathcal{G}$ and spanning graph $\mathcal{H}$ derived from a given terrain; (b) Coverage planning results using two robots starting from their individual depots (highlighting with stars). Their coverage paths are highlighted in brown and blue, respectively.

B. MSTC* Path Planning

In this section, we present the MSTC* algorithm as the inspiration for our work. Initially, MSTC* executes Kruskal's algorithm to create a minimum spanning tree (MST) on $\mathcal{H}$ by removing edges with high weights and obtaining a minimal-weighted coverage path Π on $\mathcal{G}$. An example of coverage path planning on a grid graph using Kruskal's algorithm is shown in Fig. 3(a). Subsequently, MSTC* considers coverage path planning for multiple robots as a problem of dividing a topological loop into k well-balanced partitions $\mathcal{P}$, taking

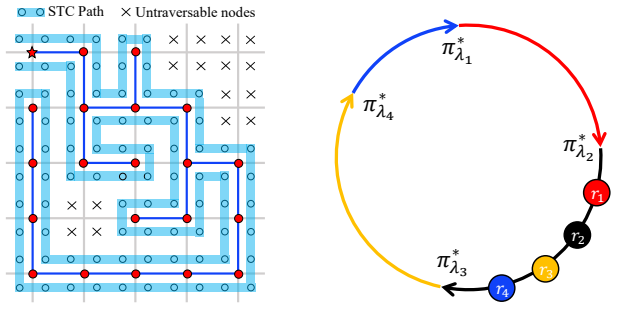
into account the cost of coverage paths, and allocating each partition to a robot (see Fig. 3(b)). The loop corresponds to the minimal-weighted coverage path Π, and k partitions of Π can be denoted by k key nodes as follows:

$$\mathcal{P} = \{\pi^*_{\lambda_1}, \pi^*_{\lambda_2}, \cdots, \pi^*_{\lambda_k}\} \tag{2}$$

π^* represents the key node that separates two adjacent partitions. λ_i is the node index from which a partition of coverage path starts. Then a partition is the coverage path Π consisting of all nodes between two key nodes.

$$\Pi_i = \{\pi^*_{\lambda_i}, \pi_{\lambda_i+1}, \pi_{\lambda_i+2}, \cdots, \pi^*_{\lambda_{i+1}-1}\} \tag{3}$$

MSTC* first generates k partitions such that each coverage path Π_i contains an equal number of covering nodes. Then, a greedy algorithm is employed to gradually shift nodes from the coverage path with the highest weight, Π_{max}, to the coverage path with the lowest weight, Π_{min}, through intermediate partitions.



(a) Kruskal STC Path (b) MSTC* Path Partition

Fig. 3: MSTC* coverage path generation and path partition strategy

MSTC* was originally designed for restoring degraded lands with complex surfaces and topologies using tree-planting robots. Thus, it incorporates ground slope and path length into the path weight. However, in flat environments, the number of turns has a greater impact on coverage time than ground slope. To balance coverage time for multiple robots and reduce the overall time required for coverage, especially in scenarios with unevenly distributed obstacles, we propose the TMSTC* algorithm, which incorporates the number of turns into the path weight.

C. Turn-minimizing mCPP Pipeline

Fig. 4 shows us that the shape of the spanning tree greatly affects the number of turns in the robot's path. A randomly generated spanning tree can increase the number of turns in the coverage task dramatically. However, turns can be reduced by constructing a spanning tree with long straight bricks (see Fig. 4-(b)). A brick B is defined as a rectangle with integer side lengths, i.e., with a width of $2d$ or a height of $2d$ (or both). A brick is also considered as a sequence of spanning nodes in $\mathcal{H}$ with the (x, y)-coordinates:

$$B_i = (\rho^i_1, \rho^i_2, \cdots, \rho^i_n) \tag{4}$$

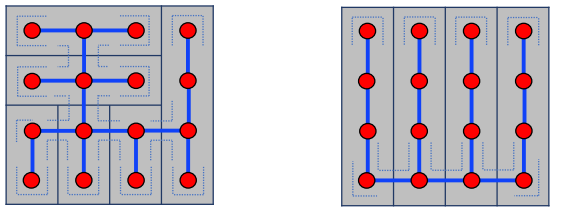
where $x_{\rho^i_1} = x_{\rho^i_2} = \cdots = x_{\rho^i_n}$ or $y_{\rho^i_1} = y_{\rho^i_2} = \cdots = y_{\rho^i_n}$, ρ^i_j and ρ^i_{j+1} are adjacent. ρ^i_j denotes the j-th element in B_i.

We aim to tile the empty region with minimum bricks, such that each non-occupied grid cell in the map is covered by exactly one brick, and each occupied grid cell is not covered by any brick. Bricks do not overlap. Since the robots are only required to turn between two connecting or adjacent bricks, we can reduce the turns by minimizing the number of bricks. The minimum brick tiling is represented as

$$\begin{aligned} &\min \|\{B_1, B_2, \cdots, B_m\}\| \\ &s.t. \quad B_1 \cup B_2 \cup \cdots \cup B_m = \mathcal{H} \\ &\qquad B_i \cap B_j = \varnothing, \quad \forall i, j \in [1, m], i \neq j \end{aligned} \tag{5}$$

After merging the bricks into a spanning tree using a greedy strategy, we use the improved MSTC* algorithm to plan a set of coverage paths $\{\Pi_i\}$ for k robots.In short, our turn-minimizing algorithm for mCPP consists of three stages.

1) Partitioning the empty region with minimum non-overlap bricks
2) Merging bricks with greedy strategy for further reduction of turns
3) Computing paths $\{\Pi_i\}$ for robots with improved MSTC* algorithm



(a) Randomly generated tree with 34 turns (b) Ideal-shaped tree with 16 turns

Fig. 4: Comparison of the number of turns in different spanning trees. Each grey rectangle represents a brick. There are 6 bricks in (a) and 4 bricks in (b).

IV. Our Algorithm

A. Minimum Brick Tiling in Grid Graphs

It is not easy to directly find a minimum brick tiling in a grid graph. However, we can observe that each pair of adjacent grid cells in the spanning graph $\mathcal{H}$ is divided by a line segment of length $2d$. Therefore, we can try to remove as many line borders between adjacent non-occupied grid cells as possible to form a minimum brick tiling. However, it is important to note that the bricks must not be L-shaped, since we have previously established that bricks must be rectangular.

We define the vertex set corresponding to the horizontal borders as H, and the vertex set corresponding to the vertical borders as V. We then construct a bipartite graph $\mathcal{I} = (\{H, V\}, E)$, where H and V are two disjoint and independent sets, and every edge $e \in E$ connects a vertex in H to one in V. An independent set is a set of vertices where no two vertices in the set are adjacent to each other. For each edge $(h, v) \in E$, the line segments corresponding to h and v are mutually perpendicular and share an endpoint. The edges in the bipartite graph are undirected. Figures 6-(a) and (b) illustrate the process of transforming a grid graph into a bipartite graph $\mathcal{I}$. We will use the following lemmas to

reduce the minimum brick tiling problem to a maximum flow problem in graph theory, and then demonstrate how to solve it using relevant algorithms.

Lemma 1. *Every brick tiling of the grid graph corresponds to an independent set in $\mathcal{I}$. Specifically, a minimum brick tiling corresponds to a maximum independent set.*

Proof. Assuming there exists a non-independent set W in $\mathcal{I}$, then there exists at least one edge (h, v) in $\mathcal{I}$ where $h, v \in W$, $h \in H$, and $v \in V$. According to the definition, two adjacent vertices in $\mathcal{I}$ correspond to two adjacent and mutually perpendicular line segments. However, deleting two segments that share an endpoint will create an L-shaped block, which conflicts with the definition of bricks. Illegal segment elimination is shown in Fig. 5-(c), while legal ones are shown in Fig. 5-(a) and (b). For any independent set IS of $\mathcal{I}$, there are no two vertices corresponding to adjacent and mutually perpendicular line segments in $\mathcal{I}$, i.e., $\forall a, b \in IS, (a, b) \notin E$. Any vertex subset of $\mathcal{I}$ is either independent or non-independent, so a brick tiling must have a one-to-one corresponding independent set in $\mathcal{I}$.

Let R, S, and T denote the number of bricks, the number of non-occupied grid cells, and the size of the independent set, respectively. If we do not eliminate any segments initially, then we will have S bricks. For each vertex added to the independent set, it represents the merging of two bricks into one, and the total number of bricks decreases by one because the line segment corresponding to that vertex is removed. Therefore, we can obtain the equation:

$$R = S - T \tag{6}$$

Minimizing R is equivalent to maximizing T since S is fixed, thus a minimum brick tiling of the grid graph corresponds to a maximum independent set of $\mathcal{I}$, specifically. □

At this point, we have transformed the minimum brick tiling problem into finding the maximum independent set of $\mathcal{I}$. Although this task is NP-hard for general graphs, we only need to solve it for the bipartite graph, which can be further reduced to finding the minimum vertex cover, a problem that can be solved in polynomial time. Formally, a vertex cover V' of an undirected graph $G = (V, E)$ is a subset of V such that if $(u, v) \in E$, then at least one of u and v belongs to V'. In other words, every edge in G is *"covered"* by at least one vertex in V'. A vertex cover of minimum size is called a minimum vertex cover of the graph.

Lemma 2. *The maximum independent set is the complement of the minimum vertex cover.*

Proof. Let X be a maximum independent set of G. For each edge $e \in E(G)$, e has at least one vertex does not belong to X, implying that $V(G)\backslash X$ is a vertex cover. If $V(G)\backslash X$ is not a minimum vertex cover, then there exists a vertex $v \in V(G)\backslash X$ such that $(V(G)\backslash X) - \{v\}$ is also a vertex cover. Thus, all vertices in the neighbourhood $N(v)$ of v must belong to $V(G)\backslash X$, that is, $\{v\} \cup N\{v\}$ is disjoint from X. Hence, $X \cup \{v\}$ is also an independent set of G, which contradicts the maximality of X. □

Furthermore, König's theorem states that in any bipartite graph, the size of a maximum matching is equal to the size of a minimum vertex cover. The proof of König's theorem in [21] also provides an algorithm for finding a minimum vertex cover of a bipartite graph in polynomial time using a max flow algorithm. Algorithm 1 provides the pseudocode to find the minimum vertex cover. The flow network $\mathcal{F} = (\mathcal{I}', s, t, c)$ corresponding to $\mathcal{I} = (\{H, V\}, E)$ mentioned on line 2 is defined as follows:

- s and t are new nodes not in $H \cup V$
- $\mathcal{I}' = (U, E')$ is the directed graph with set of vertices $U = H \cup V \cup \{s, t\}$ and set of edges $E' = \{(s, h) : h \in H\} \cup \{\{h, v\} : h \in H, v \in V, \{h, v\} \in E\} \cup \{(v, t) : v \in V\}$
- the capacity $c(e)$ for each $e \in E'$ is equal to 1

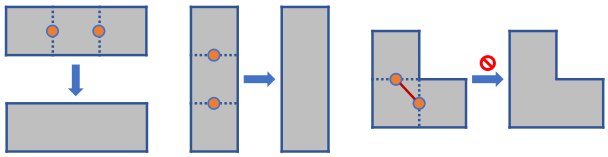


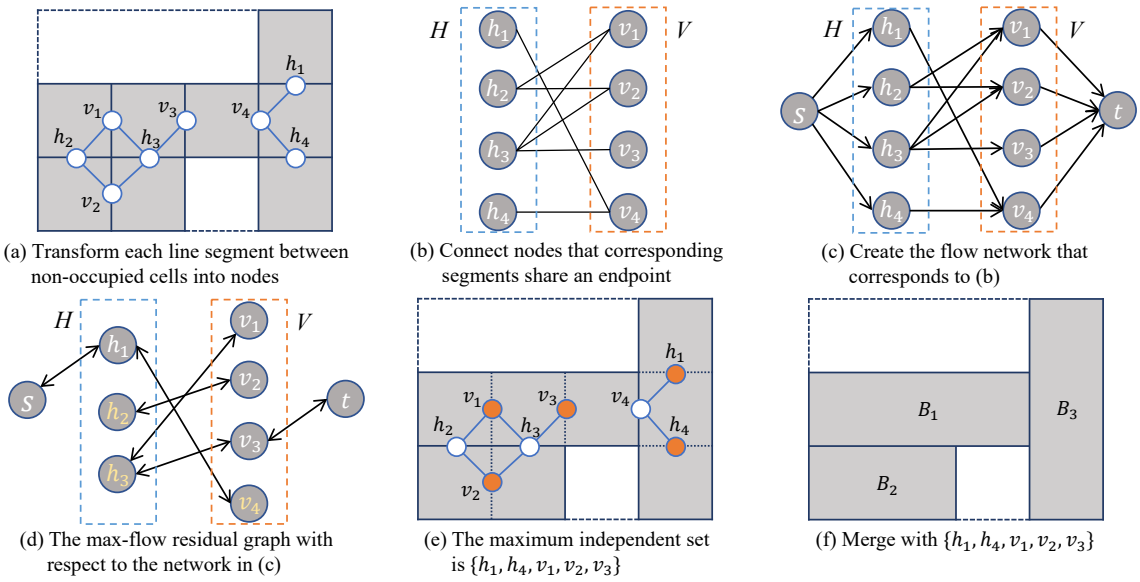
Fig. 5: Example of line segments elimination.

After obtaining a minimum vertex cover, we can obtain the maximum independent set required by taking its complement. Then, we can derive the minimum brick tiling of $\mathcal{H}$ by removing the corresponding segments from the maximum independent set. In summary, we solve the minimum brick tiling problem by converting it to a maximum flow problem in a flow network. To accomplish this, we use Dinic's algorithm, which has a time complexity of $O(\sqrt{V}E)$ in bipartite graphs, where V and E are the sizes of the vertices and edges in the bipartite graph, respectively [22].

Fig. 6 illustrates an example of using our algorithm to find a minimum brick tiling of a 3×4 regular grid graph. Each line border between two free cells in Fig. 6-(a) represents a node in bipartite graph $\mathcal{I}$ shown in Fig. 6-(b). We put an edge between two nodes in $\mathcal{I}$ when their corresponding segments share the

Algorithm 1: Bipartite Minimum Vertex Cover

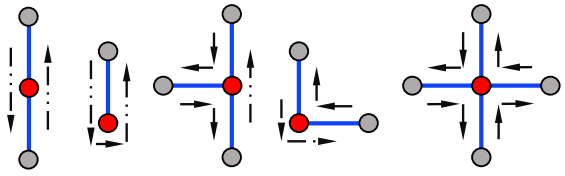
Input: spanning graph $\mathcal{H}$
Output: minimum vertex cover
1 $\mathcal{I}$:= the bipartite graph corresponds to $\mathcal{H}$
2 $\mathcal{F}$:= the flow network that corresponds to $\mathcal{I}$
3 f := `MaxFlow`($\mathcal{F}$)
4 $\mathcal{F}_f$:= the residual graph of flow f with respect to network $\mathcal{F}$
5 $C := \{c : c$ is a node not reachable from s in $\mathcal{F}_f$ and $c \in H\}$
6 $D := \{d : d$ is a node reachable from s in $\mathcal{F}_f$ and $d \in V\}$
7 **return** $C \cup D$



(a) Transform each line segment between non-occupied cells into nodes
(b) Connect nodes that corresponding segments share an endpoint
(c) Create the flow network that corresponds to (b)
(d) The max-flow residual graph with respect to the network in (c)
(e) The maximum independent set is $\{h_1, h_4, v_1, v_2, v_3\}$
(f) Merge with $\{h_1, h_4, v_1, v_2, v_3\}$

Fig. 6: Example of finding a minimum brick tiling of a grid graph.

same endpoint. Figs. 6-(c) and (d) demonstrate the process of using Algorithm 1 to find a minimum vertex cover of $\mathcal{I}$. The minimum vertex cover is $\{h_2, h_3, v_4\}$ since h_2 and h_3 are not reachable from s in the residual network (where h_2 and h_3 are in H), but v_4 is (which is in V). The complement of this set, $\{h_1, h_4, v_1, v_2, v_3\}$, is the maximum independent set. Figs. 6-(e) and (f) demonstrate that using set $\{h_1, h_4, v_1, v_2, v_3\}$ to merge cells leads to three bricks, which is the minimum brick tiling in this example.



(a) I-shaped (b) Leaf (c) T-shaped (d) L-shaped (e) +-shaped

Fig. 7: Five different types of nodes in spanning graph $\mathcal{H}$. The dotted black arrow indicates the robot's trajectory around the node.

B. Combining Bricks Into A Spanning Tree

After having least number of bricks as spanning tree branches, we want to merge them for constructing a tree containing all free cells with the rest of unused edges. When robots move along the spanning tree, they produce different number of turns at various kinds of vertex. Fig. 7 shows five kinds of nodes with various turns. Let function $f(\rho)$ represent the number of turns around tree vertex ρ.

$$f(\rho) = \begin{cases} 0 & deg(\rho) = 2,\ \rho \text{ and its two neighbours are in the same line} \\ 2 & deg(\rho) = 2,\ \rho \text{ and its two neighbours are not in the same line} \\ 2 & deg(\rho) = 1 \text{ or } deg(\rho) = 3 \\ 4 & deg(\rho) = 4 \end{cases} \tag{7}$$

Here, $deg(\rho)$ represents the degree of vertex ρ in the spanning tree, which is the number of edges that are incident to the vertex.

Each time a new edge is added, the number of degrees and connections of the vertices change, which in turn affects the number of turns. We define the function $g(e_{\rho_i,\rho_j})$ as the cost of adding a new edge. It would be

$$g(e_{\rho_i,\rho_j}) = f^+(\rho_i) + f^+(\rho_j) - f(\rho_i) - f(\rho_j) \tag{8}$$

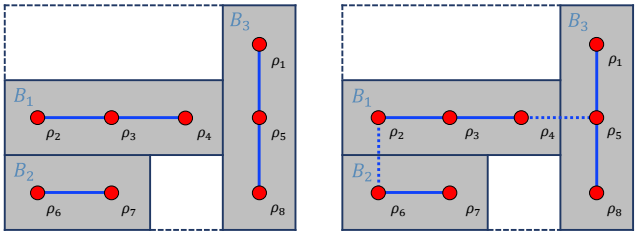
where $f^+(\rho_i)$, $f^+(\rho_i)$ represent turns of ρ_i, ρ_j after adding edge e_{ρ_i,ρ_j} respectively.

It is important to note that adding a new edge will only affect the number of turns for the two vertices connected by that edge. The function $g(e)$ measures the increase in the total number of turns resulting from adding the new edge e. To construct a spanning tree in a step-by-step manner, we adopt a greedy strategy where we choose the edge with the lowest cost at each step and use it to merge bricks. Algorithm 2 summarizes the pseudo-code for this strategy.

Fig. 8 provides an example of how to build a spanning tree. First, we connect adjacent nodes within each brick, and then we add two more edges to connect all three bricks and form a complete spanning tree. There are three edges to choose from: e_{ρ_2,ρ_6}, e_{ρ_3,ρ_7}, and e_{ρ_4,ρ_5}, with corresponding costs of 0, 2, and 0, respectively. We select the lowest-cost edge, e_{ρ_2,ρ_6}, to connect B_1 and B_2. Connecting B_1 and B_2 makes ρ_3 and ρ_7 no longer disjoint, so we discard e_{ρ_3,ρ_7}. Finally, we use e_{ρ_4,ρ_5} to connect B_1 and B_3. This results in a complete spanning tree that minimizes turns.

C. Improved MSTC With Turning Cost*

The original MSTC* algorithm assumes that the robot only moves uniformly while covering and does not consider the cost of turning, which can slow down the robot and increase its energy consumption. However, in practice, it is essential to balance the coverage time with the cost of turning, in addition



(a) Bricks before merging (b) Merge bricks to form a spanning tree

Fig. 8: Merging bricks with the rest of edges in spaning graph $\mathcal{H}$. The dotted lines denotes edges connect different bricks.

Algorithm 2: Merge Bricks

Input: vertex set of bricks $\{B_1, B_2, ..., B_m\}$
Output: edge set T of spanning tree in graph $\mathcal{H}$

$T := \varnothing$
$m :=$ the number of bricks
for $i := 1$ **to** m **do**
  $E_i :=$ the set of edges connecting adjacent nodes in brick B_i
  $T := T \cup E_i$
end
while $m > 1$ **do**
  $e_{\rho_i,\rho_j} :=$ the edge with the smallest cost $g(\cdot)$ that connects adjacent nodes ρ_i and ρ_j, where ρ_i and ρ_j are disjoint in T
  $T := T \cup e_{\rho_i,\rho_j}$
  $m := m - 1$
end
return T

to the path length. To achieve this, we need to incorporate the cost of turns into the robot's accumulating cost $\mathcal{W}_{\Pi_i}$. Let a, v_{max}, and ω denote the robot's acceleration, maximum linear velocity, and angular velocity, respectively. We can represent a 90° turn as a twist γ. Each coverage path Π_i has a corresponding set of twists Γ_i.

$$\Gamma_i = \{\gamma_1, \gamma_2, \cdots, \gamma_n\} \tag{9}$$

γ_1 and γ_n denote start and end points of the coverage path Π_i, respectively. Fig. 9 gives an example of twists in a robot covering path.

To simplify the problem, we make the assumption that robots perform uniformly accelerated motion and stop at every twist in order to rotate during the entire coverage task. Based on the path, we can calculate the amount of time T_i required by each robot to complete the coverage of the grid cells using the following formula:

$$T_i = \sum_{j=1}^{n-1} t_j + \frac{(n-2)\pi}{2\omega}$$

$$t_j = \begin{cases} \sqrt{\frac{d_{\gamma_j,\gamma_{j+1}}}{a}} & d_{\gamma_j,\gamma_{j+1}} \leq \frac{v_{max}^2}{a} \\ \frac{d_{\gamma_j,\gamma_{j+1}}}{v_{max}} + \frac{v_{max}}{a} & \text{otherwise} \end{cases} \tag{10}$$

where $d_{\gamma_j,\gamma_{j+1}}$ and t_j denote euclidean distance and covering time, respectively, between γ_j and γ_{j+1}. There are two cases for calculating t_j based on the previous assumptions: if the distance between two twists is less than $\frac{v_{max}^2}{a}$, the robot must decelerate to 0 to reach the next twist without having accelerated to v_{max}, or the robot accelerates to v_{max} and then decelerates to 0 after a certain time to reach the next twist. The robot will make $(n-2)$ turns on path Π_i, and the required time for turning is $\frac{(n-2)\pi}{2\omega}$. We use covering time T_i as the cost of the path $\mathcal{W}_{\Pi_i}$ in our implementation.

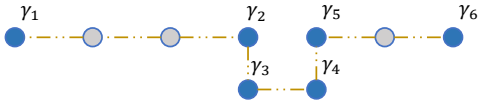


Fig. 9: Example of twists (highlighted in blue) in a coverage path. The robot starts at γ_1 and stops at γ_6, making a total of four turns during the coverage.

V. Results & Analysis

In this section, we conduct experiments to test the performance of TMSTC* and compare it with MSTC*, TMC, and DARP. Our experimental validation aims to demonstrate the differences in the number of turns, coverage time, and computation time of these algorithms under different environments. TMC simplifies the environment into rectilinear polygons and divides them into checkerboard-like rectangles with varying sizes and orientations. Iterative heuristics are used to identify the minimum number of coverage lines required for complete coverage, which are then allocated to multiple robots using m-TSP. DARP, on the other hand, uses gradient descent to divide the environment into equal and exclusive sub-regions for each robot, based on their initial positions. Each robot operates the spanning tree coverage algorithm within its respective sub-region to achieve complete coverage. Since TMC employs a different map partitioning approach, we conducted experiments where it covered the same area as TMSTC*, MSTC*, and DARP.

We implement candidate algorithms compiled by C++ on ROS Melodic and use Gazebo with TurtleBot kobuki model for physics simulation. Table I shows the kobuki kinematic parameters. We have created a regular indoor map and an outdoor map with random obstacles as benchmark environments. In terms of the performance of large-scale mCPP problems, we additionally test candidate algorithms in two street maps NewYork_0 and Denver_2, which are discretized from building/road maps from Real-World Benchmarks [23]. These environments appear in Fig. 12-(a)∼(d), where dark areas indicate obstacles and white areas the space to cover.

TABLE I: Robot parameters used for simulations

Coverage Tool Width	$0.5m$
Maximum Translational Velocity	$0.5m/s$
Maximum Rotational Velocity	$0.8rad/s$
Translational Acceleration	$0.6m/s^2$

A. Turn comparisons

In multi-robot coverage, each robot may be assigned a different number of turns. To evaluate the algorithms' performance in terms of turns, we first compare the overall number

of turns in single-robot scenarios. MSTC* uses Kruskal to generate STC paths, while DARP uses DFS. We also compared the number of turns of the STC paths generated by BFS and Ant Colony Optimization (ACO). The results are presented in Table II. Our proposed method generates paths with significantly fewer turns in all four maps, indicating its superiority in reducing turns in various environments. On the other hand, BFS and DFS perform poorly in all four maps as they lack a strategy to reduce turns. Although ACO and TMC use different random heuristic strategies to reduce the number of turns, they are prone to getting trapped in local optima for more complex and unstructured, irregular-shaped large-scale terrains, leading to suboptimal results. Kruskal, while not guaranteed to generate fewer turns, outperforms ACO and DFS overall. However, it still lacks flexibility in tight spaces, resulting in unnecessary turns.

TABLE II: The number of turns

Terrain	Ours	TMC	BFS	DARP	ACO	MSTC*
Indoor	**206**	260	326	296	254	272
Random	**208**	228	322	296	288	258
Denver_2	**652**	878	1024	1000	1076	894
NewYork_0	**780**	976	1176	1180	1440	954
Total	**1846**	2342	2848	2772	3058	2378

B. Coverage Time Comparisons

During our experiments, we made certain adjustments to the parameters of TMC and DARP. Specifically, we set the maximum iterations for TMC's heuristic method to 5000 in order to locate the minimum coverage lines. For the DARP algorithm, we set the maximum number of iterations to 50000 and ensured that the robots were evenly distributed throughout the configuration space to allow for sufficient iterations. Each algorithm was tested with five different numbers of robots for each map. For indoor and outdoor maps, we evaluated the algorithms with 1, 2, 3, 4, and 5 robots, while for Denver_2 and NewYork_0, we tested the algorithms with 1, 4, 8, 12, and 16 robots.

In the context of multi-robot coverage, it is crucial to consider that the completion time of the task is determined by the time taken for the last robot to finish its coverage. To evaluate the efficiency of our algorithm in comparison to existing methods, we have graphed the percentage improvement of TMSTC* in completion time relative to MSTC*, TMC, and DARP in Fig. 10-(a), (b), and (c), respectively. Additionally, we have calculated the sum of the time differences between the last and first robots to finish the coverage for each algorithm under varying numbers of robots on each map to measure load balancing. The results are plotted in Fig. 10-(d). We have also included the planning path of TMSTC* displayed on rviz in Fig. 12-(a)∼(d).

After comparing the coverage time among the four algorithms, we found that TMSTC* outperforms the other methods in all scenarios, achieving a faster coverage time by an average of 6.4%, 9.2%, and 12.5% compared to TMC, MSTC*, and DARP, respectively, across all test cases. This indicates that TMSTC* is more efficient than existing methods due to its ability to reduce the number of turns and incorporate the cost of turns into path partition. As a result, the time difference between the first and last robot to complete coverage is much smaller compared to the other algorithms.

Upon observing the maps of Denver_2 and NewYork_0, it becomes apparent that certain areas on the map have a higher number of turns compared to others. TMC and MSTC* assign coverage paths of similar lengths to robots, which can result in an uneven distribution of turns among them. In such situations, TMSTC* can even achieve a reduction in completion time of over 10% compared to these two algorithms. On the other hand, the DARP algorithm divides areas into equally sized regions with random shapes, ignoring the number of turns. As a result, this approach leads to an unequal coverage time among robots and increases the total time required to complete the task. In conclusion, our results suggest that TMSTC* outperforms other algorithms in scenarios with complex maps where the number of turns is imbalanced across different regions.

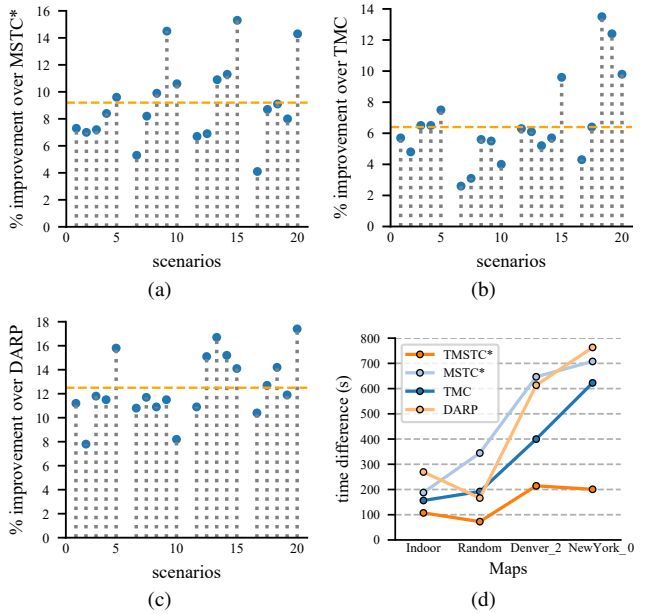


Fig. 10: Comparisons of the performance of TMSTC* with the other three algorithms. (a)∼(c) Average percentage improvement achieved by TMSTC* compared to MSTC*, TMC, and DARP in each scenario. (d) A comparison of the sum of time differences between the last and first robots to complete coverage for the four algorithms on each map.

C. Computation Time Comparisons

We performed a statistical analysis of the total computation time for each algorithm on five different scenarios per map, as shown in Figure 11. The planning time of TMSTC* is very similar to that of MSTC*, indicating that our improvement methods are efficient. However, the path partitioning step in the original MSTC* algorithm accounts for the majority of the planning time. Compared to TMSTC* and MSTC*, TMC takes more planning time because it adopts an asymptotically partitioning heuristic that requires repeated random searches to obtain a relatively optimal solution. While DARP is faster than TMSTC*, MSTC*, and TMC on small maps, it is much slower on larger ones due to the difficulty in converging using

gradient descent for equally-sized area allocation. In summary, TMSTC* can plan paths with fewer turns and more balanced coverage in less time for multiple robots.

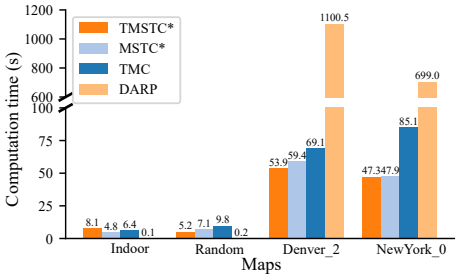
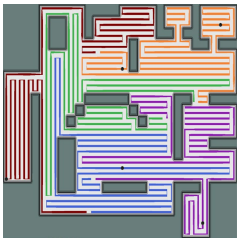
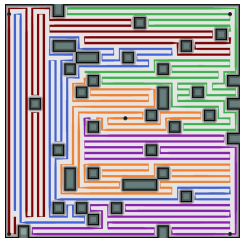


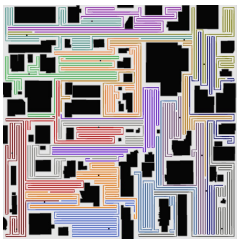
Fig. 11: Computation time Comparisons.



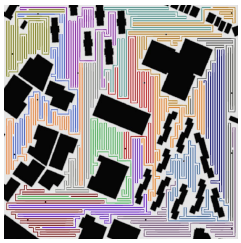
(a) Regular indoor map with 52×42 cells



(b) Random obstacle map with 40×40 cells



(c) Denver_2 with 108×108 cells



(d) NewYork_0 with 108×108 cells

Fig. 12: Four different environments. (a)∼(d) The corresponding results of coverage path planning using our TMSTC* algorithm. We apply 5 robots in (a)(b) and 16 robots in (c)(d).

VI. Conclusions & Future Work

In this paper, we propose the TMSTC* coverage planning algorithm, which aims to minimize the number of turns and reduce coverage time. Our algorithm improves on MSTC* by constructing a tree that allows robots to make fewer turns when circumnavigating, and incorporates turning cost when balancing weights. Comparative experiments with the MSTC*, TMC, and DARP algorithms show that TMSTC* has advantages in terms of planning time for both simple and complex configuration spaces.

Fault tolerance is vital for multi-robot frameworks, especially for those operating in remote environments. For future work, we plan to extend TMSTC* to tolerate failures and replan trajectories for functional robots without immediate human intervention.

References

[1] Enric Galceran and Marc Carreras. A survey on coverage path planning for robotics. *Robotics and Autonomous systems*, 61(12):1258–1276, 2013.

[2] Jaeseok Kim, Anand Kumar Mishra, Raffaele Limosani, Marco Scafuro, Nino Cauli, Jose Santos-Victor, Barbara Mazzolai, and Filippo Cavallo. Control strategies for cleaning robots in domestic applications: A comprehensive review. *International Journal of Advanced Robotic Systems*, 16(4):1729881419857432, 2019.

[3] Gert Kootstra, Xin Wang, Pieter M Blok, Jochen Hemming, and Eldert Van Henten. Selective harvesting robotics: current research, trends, and future directions. *Current Robotics Reports*, 2:95–104, 2021.

[4] Li Huang, MengChu Zhou, Kuangrong Hao, and Edwin Hou. A survey of multi-robot regular and adversarial patrolling. *IEEE/CAA Journal of Automatica Sinica*, 6(4):894–903, 2019.

[5] Fronefield Crawford, Tim Bechtel, Gennadiy Pochanin, Pierluigi Falorni, Khaled Asfar, Lorenzo Capineri, and Mattia Dimitri. Demining robots: Overview and mission strategy for landmine identification in the field. In *2021 11th International Workshop on Advanced Ground Penetrating Radar (IWAGPR)*, pages 1–4. IEEE, 2021.

[6] Randa Almadhoun, Tarek Taha, Lakmal Seneviratne, and Yahya Zweiri. A survey on multi-robot coverage path planning for model reconstruction and mapping. *SN Applied Sciences*, 1(8):1–24, 2019.

[7] Esther M Arkin, Michael A Bender, Erik D Demaine, Sándor P Fekete, Joseph SB Mitchell, and Saurabh Sethia. Optimal covering tours with turn costs. *SIAM Journal on Computing*, 35(3):531–566, 2005.

[8] Jingtao Tang, Chun Sun, and Xinyu Zhang. Mstc*: Multi-robot coverage path planning under physical constrain. In *2021 IEEE International Conference on Robotics and Automation (ICRA)*, pages 2518–2524. IEEE, 2021.

[9] Isaac Vandermeulen, Roderich Groß, and Andreas Kolling. Turn-minimizing multirobot coverage. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 1014–1020. IEEE, 2019.

[10] Howie Choset and Philippe Pignon. Coverage path planning: The boustrophedon cellular decomposition. In *Field and service robotics*, pages 203–209. Springer, 1998.

[11] Stanislav Bochkarev and Stephen L Smith. On minimizing turns in robot coverage path planning. In *2016 IEEE international conference on automation science and engineering (CASE)*, pages 1237–1242. IEEE, 2016.

[12] Rik Bähnemann, Nicholas Lawrance, Jen Jen Chung, Michael Pantic, Roland Siegwart, and Juan Nieto. Revisiting boustrophedon coverage path planning as a generalized traveling salesman problem. In *Field and Service Robotics: Results of the 12th International Conference*, pages 277–290. Springer, 2021.

[13] Jian Jin and Lie Tang. Optimal coverage path planning for arable farming on 2d surfaces. *Transactions of the ASABE*, 53(1):283–295, 2010.

[14] Megnath Ramesh, Frank Imeson, Baris Fidan, and Stephen L Smith. Optimal partitioning of non-convex environments for minimum turn coverage planning. *IEEE Robotics and Automation Letters*, 7(4):9731–9738, 2022.

[15] Xinyue Kan, Hanzhe Teng, and Konstantinos Karydis. Online exploration and coverage planning in unknown obstacle-cluttered environments. *IEEE Robotics and Automation Letters*, 5(4):5969–5976, 2020.

[16] Yoav Gabriely and Elon Rimon. Spanning-tree based coverage of continuous areas by a mobile robot. *Annals of mathematics and artificial intelligence*, 31(1):77–98, 2001.

[17] Noa Agmon, Noam Hazon, and Gal A Kaminka. Constructing spanning trees for efficient multi-robot coverage. In *Proceedings 2006 IEEE International Conference on Robotics and Automation, 2006. ICRA 2006.*, pages 1698–1703. IEEE, 2006.

[18] Xiaoming Zheng, Sven Koenig, David Kempe, and Sonal Jain. Multirobot forest coverage for weighted and unweighted terrain. *IEEE Transactions on Robotics*, 26(6):1018–1031, 2010.

[19] Athanasios Ch Kapoutsis, Savvas A Chatzichristofis, and Elias B Kosmatopoulos. Darp: divide areas algorithm for optimal multi-robot coverage path planning. *Journal of Intelligent & Robotic Systems*, 86(3):663–680, 2017.

[20] Savvas D Apostolidis, Pavlos Ch Kapoutsis, Athanasios Ch Kapoutsis, and Elias B Kosmatopoulos. Cooperative multi-uav coverage mission planning platform for remote sensing applications. *Autonomous Robots*, 46(2):373–400, 2022.

[21] John Adrian Bondy, Uppaluri Siva Ramachandra Murty, et al. *Graph theory with applications*, volume 290. Macmillan London, 1976.

[22] Yefim A Dinitz. An algorithm for the solution of the problem of maximal flow in a network with power estimation. In *Doklady Akademii nauk*, volume 194, pages 754–757. Russian Academy of Sciences, 1970.

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