Minimum-Length Coverage Path Planning for Grid Environments with Approximation Guarantees

Megnath Ramesh[1], Frank Imeson[2], Baris Fidan[1], and Stephen L. Smith[1]

***Abstract*—We focus on planning minimum-length robot paths to cover environments using the robot's sensor or coverage (e.g., cleaning) tool. Many algorithms use the following framework: (i) compute a grid decomposition of the environment, (ii) partition the grid to be covered by non-overlapping *coverage lines* (straight-line paths), and (iii) compute a cost-minimizing tour of the coverage lines to get a coverage path. While this framework aims to minimize turns in the path, it does not yield guarantees on the resulting path length. In this paper, we show that this framework guarantees a coverage path of length $(1+1.5\gamma)$ times the optimal, where $\gamma > 1$ is the approximation factor to solve the metric traveling salesman problem (metric-TSP). Following this, we propose the Minimum Length Coverage Approx (MLC-Approx) approach that modifies this framework to achieve an approximation factor of $(1.5+\epsilon)$, where $\epsilon \ll 1$ depends on the number of coverage lines. Instead of computing a tour of the coverage lines, MLC-Approx merges minimum-length *sub-tours* of coverage lines while minimizing the turns added by the merges. We also propose a lazy variation of MLC-Approx that achieves the same result with faster empirical runtime. We validate MLC-Approx in simulations using maps of real-world environments and compare against state-of-the-art CPP approaches.**

***Index Terms*—Motion and Path Planning, Service Robotics, Optimization and Optimal Control**

I. Introduction

COVERAGE path planning (CPP) is the task of planning a robot's path such that it covers its environment using a coverage tool (e.g., vacuum scrubber, sensor, plant seeder) while minimizing the cost of the path (e.g., length, turns, coverage time). While CPP can be done *online* (i.e., path planning during coverage), offline approaches are effective when the environment is mostly known and/or the path requires human verification before execution. Some applications include cleaning [1], agriculture [2], and aerial surveys [3]. However, CPP for general environments is NP-hard [4], thus optimal offline approaches do not scale well with increasing problem complexity. So, CPP algorithms in the literature [5]–[8] aim to reduce the solution complexity with the following steps: (i) decompose the environment into sub-regions, (ii) compute an optimal set of parallel straight-line paths or *coverage lines* covering each sub-region, and (iii) compute a cost-minimizing tour of the coverage lines to get a coverage path. We refer to this approach as the *line-touring framework*, which is commonly used to minimize turns in the coverage path as (i) turns take time to execute while adding little to no new coverage, and (ii) tools like cleaning attachments miss coverage during turns [6]–[9].

While this framework makes the problem more tractable, it, in general, computes paths with suboptimal length. This is acceptable as long as the computed solution is "good enough" with a known approximation factor. While approximation algorithms for CPP have been studied [4], [10], an approximation factor for the line-touring framework has not been analyzed. In this work, we study the line-touring framework applied to grid decompositions, where the environment is represented as a set of square grid cells. Each coverage line covers a *rank*, i.e., a group of grid cells aligned along an axis-parallel (horizontal/vertical) straight line. As such, one can obtain the coverage lines for a grid decomposition by partitioning it into these ranks. We refer to this variation of the line-touring framework as *rank-touring* [7], [8], [10].

In this paper, we show that the rank-touring framework achieves an approximation factor of $(1 + 1.5\gamma)$ for path length, where $\gamma > 1$ is the approximation factor to solve the metric traveling salesman problem (metric-TSP). Following this, we propose a $(1.5 + \epsilon)$-approximation algorithm, namely Minimum Length Coverage Approx (MLC-Approx), for minimum-length CPP in connected grid environments with holes. Here, $\epsilon \ll 1$ is a small term proportional to the number of ranks. Building upon the rank-touring framework, MLC-Approx relaxes the turn minimization objective to compute paths with near-optimal length, while still achieving fewer turns than a TSP planner.

Contributions: Our specific contributions are as follows:

1) We show that the rank-touring framework for minimum-length CPP achieves an approximation factor of $(1 + 1.5\gamma)$, where $\gamma > 1$ is the approximation factor to solve metric-TSP,
2) For grid environments with holes, we propose MLC-Approx, a $(1.5 + \epsilon)$-approximation algorithm, for minimum-length CPP, which also minimizes the number of turns as a secondary objective,
3) We propose a lazy variation of MLC-Approx that achieves the same result with improved runtime.

We also evaluate MLC-Approx using maps of real-world environments and compare against state-of-the-art approaches that employ the rank-touring [7], [8], and line-touring [11] frameworks.

Manuscript received: April 29, 2025; Revised July 23, 2025; Accepted August 6, 2025. This paper was recommended for publication by Editor Lucia Pallottino upon evaluation of the Associate Editor and Reviewers' comments. This work was supported by Canadian Mitacs Accelerate Projects IT16435 and IT40666, and Avidbots Corp, Kitchener, ON, Canada.

[1]M. Ramesh (m5ramesh@uwaterloo.ca) and S. L. Smith (stephen.smith@uwaterloo.ca) are with the Department of Electrical and Computer Engineering and B. Fidan (fidan@uwaterloo.ca) is with the Department of Mechanical and Mechatronics Engineering, at the University of Waterloo, Waterloo ON, Canada

[2]F. Imeson (frank.imeson@avidbots.com) is with Avidbots Corp., Kitchener ON, Canada

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

Related Works: Detailed surveys of CPP approaches can be found in [12], [13]. Broadly, CPP algorithms are classified into (i) *exact* and (ii) *approximate* approaches. *Exact* CPP approaches first decompose the environment into (usually convex) cells, and then use line-touring to compute a coverage path [6], [11]. Boustrophedon [5] is a widely-used approach of this category, where the coverage lines for each cell are oriented along a single horizontal orientation. Variations of boustrophedon focus on minimizing turns in the coverage path by covering each cell with the minimum number of coverage lines [14], [15]. However, for environments with complex boundaries, these approaches suffer from *over-decomposition*, i.e., decompositions containing narrow cells with widths smaller than the coverage tool, resulting in overlapping paths and increased path length. These approaches also do not provide guarantees on the resulting path length.

Approximate approaches are another category of CPP algorithms, which aim to cover a subset of the environment that the coverage tool can realistically reach. This includes *grid-based* CPP approaches, where the environment is decomposed into (usually square) grid cells and the coverage path must cover each grid cell. Some grid-based approaches also provide optimality guarantees on the coverage path length [4], [9]. One such approach is spanning-tree coverage (STC), which computes minimum-length coverage paths for a square coverage tool of width l covering a grid environment with square cells of width $2l$ [16]. However, these guarantees do not directly apply to general grid environments, i.e., cells of width l. More recently, a turn-minimizing variant of STC was introduced [17], albeit with the same drawback. In this work, we propose an approximation algorithm for general grid environments that also aims to minimize the number of turns in the path.

Recent turn-minimizing approaches apply the rank-touring framework to minimize turns in the coverage path [7], [8]. In [8], the authors proposed the optimal axis-parallel rank partitioning (OARP) approach that guarantees the minimum number of axis-parallel ranks in the coverage path. In this work, we provide an approximation factor for these approaches and propose an algorithm with a tighter guarantee.

The above CPP approaches compute paths *offline* by considering the environment to be known. In contrast, *online* CPP approaches [18]–[20] assume that the environment is partially/fully unknown and plan paths during coverage. However, as these approaches often plan paths without complete environment information, they tend not to minimize the path cost effectively compared to offline approaches. In our work, we propose a lazy offline approach with fast runtimes for potential usage in online settings.

Organization: In Section II, we define CPP and the minimum-length rank-based CPP problem that we look to solve. In Section III, we establish an approximation factor of $(1 + 1.5\gamma)$ for the rank-touring framework to solve minimum-length CPP. In Section IV, we propose MLC-Approx and show that it achieves an improved approximation factor of $(1.5 + \epsilon)$, where $\epsilon \ll 1$. In Section V, we introduce a lazy variation of MLC-Approx to improve CPP computation time. In Section VI, we provide simulation results where we compare MLC-Approx with baseline approaches that follow the same framework. Section VII concludes the paper.

II. Problem Definition

A. Coverage Planning Problem (CPP)

Consider a 2D non-convex environment represented by a closed and bounded set $\mathcal{W} \subseteq \mathbb{R}^2$ with known obstacles. We consider a robot operating in $\mathcal{W}$ whose state is composed of its position $\boldsymbol{x} \in \mathcal{W}$, and heading $\theta \in [0, 2\pi)$. The robot carries a coverage tool with a footprint given by $\mathcal{A} \subseteq \mathbb{R}^2$. Let $\mathcal{A}(\boldsymbol{x}, \theta) \subseteq \mathcal{W}$ represent the placement of the tool corresponding to the robot's state. Like other works in the literature [4], [7], [17], we consider the coverage tool to be a square of width l, centered at $\boldsymbol{x}$ and oriented along θ. In reality, the square tool may not be able to cover $\mathcal{W}$ entirely. So, we look to cover an approximation $\widetilde{\mathcal{W}} \subseteq \mathcal{W}$ that the tool can reach. We consider the approximation $\widetilde{\mathcal{W}}$ to be an integral orthogonal polygon (IOP) representation of $\mathcal{W}$, which is simply a grid approximation of $\mathcal{W}$ containing square grid cells of width l. Our goal is to compute a cost-minimizing coverage path for the IOP $\widetilde{\mathcal{W}}$. We consider the coverage path to be *closed*, i.e., starts and ends at the same point. Let $\mathcal{P}$ be the set of all closed paths in $\widetilde{\mathcal{W}}$, where each $P \in \mathcal{P}$ is a trajectory passing through a sequence of robot states $(\boldsymbol{x}, \theta)$.

Problem 1 (Coverage Planning Problem)**.** *Given an approximation $\widetilde{\mathcal{W}}$ of the environment $\mathcal{W}$ and a coverage tool of width l, compute a closed coverage path $P \in \mathcal{P}$ that solves:*

$$\begin{aligned} \min_{P \in \mathcal{P}} \quad & J(P) \\ \textit{s.t.} \quad & \bigcup_{(\boldsymbol{x},\theta) \in P} \mathcal{A}(\boldsymbol{x}, \theta) = \widetilde{\mathcal{W}}, \end{aligned}$$

where $J(P)$ is the cost of the path P.

B. Rank-touring Framework

Variations of the rank-touring framework for grid decompositions are prevalent in CPP literature, especially for minimum-turn coverage planning of IOPs [7]–[9].

The rank-touring framework starts by partitioning the IOP $\widetilde{\mathcal{W}}$ into non-overlapping axis-parallel (i.e., horizontal/vertical) *ranks* $R = \{r_1, r_2, \ldots, r_m\}$, where each rank $r_i \in R$ is a set of grid cells that can be covered in a straight-line sweep using a square tool of width l. Let P_R be the set of coverage lines that each correspond to a rank in R. For example, the coverage line of a horizontal rank r_i is a path connecting the centers of its leftmost and rightmost grid cells. The framework then computes a cost-minimizing tour of the paths in P_R to obtain a coverage path for the IOP. The resulting coverage path P is a sequence of coverage lines in P_R and collision-free *transition paths* connecting two coverage lines at their endpoints. The tour aims to minimize the cost $J(P)$ associated with the transition paths.

C. Rank-based Coverage with Minimum Length

For this work, we consider the primary cost $J(P)$ of the coverage path P to be the path length $L(P)$, with the number of turns $\Theta(P)$ as a secondary cost. To minimize $\Theta(P)$, we

partition $\widetilde{\mathcal{W}}$ into axis-parallel ranks R such that the number of ranks m is minimized. We refer to this partition as the *minimum rank partition* of $\widetilde{\mathcal{W}}$.

The minimum rank partition of $\widetilde{\mathcal{W}}$ can be computed using approaches in the literature [7], [8]. From this partition, one can obtain the coverage lines P_R and a coverage path P for $\widetilde{\mathcal{W}}$ using the framework. In this work, we will focus on minimizing the length $L(P)$ of this path resulting from a minimum rank partition R and develop an approximation scheme for this problem.

Problem 2 (Minimum-length Rank-based Coverage)**.** *Given an approximation $\widetilde{\mathcal{W}}$ of the environment $\mathcal{W}$, a coverage tool of width l and a minimum rank partition R of $\widetilde{\mathcal{W}}$, compute a coverage path $P \in \mathcal{P}$ that solves:*

$$\min_{P \in \mathcal{P}} \quad L(P)$$
$$\text{s.t.} \quad \bigcup_{(\boldsymbol{x},\theta) \in P} \mathcal{A}(\boldsymbol{x},\theta) = \widetilde{\mathcal{W}}.$$

Unlike the rank-touring framework, our solution approach to this problem allows changes to the ranks (e.g., breaking) to compute near-optimal solutions in the search space. Regarding the number of turns $\Theta(P)$, we focus on rank changes that add the fewest turns to the resulting P. Note that using the minimum rank partition R also helps to minimize $\Theta(P)$ as it reduces the number of coverage lines.

III. Approximation Factor for Rank-Touring

We first analyze the optimality of the rank-touring framework to plan minimum-length coverage paths for the environment's IOP $\widetilde{\mathcal{W}}$ (Problem 2). Since it is a step-wise approach, the framework may remove optimal paths from its solution space. So, we obtain an approximation factor that quantifies the length of the resulting path in comparison to the optimal coverage path length for $\mathcal{W}$.

Given a coverage tool of width l, let n be the number of grid cells in $\widetilde{\mathcal{W}}$, and $R = \{r_1, \ldots, r_m\}$ be any rank partition of $\widetilde{\mathcal{W}}$ (does not necessarily minimize m). Each $r_i \in R$ is a horizontal/vertical rank that contains $n_i \leq n$ grid cells. Since R is a partition, $\sum_{i=1}^{m} n_i = n$. Let $P_R = \{p_1, \ldots, p_m\}$ be the set of coverage lines obtained from the ranks R, where $p_i \in P_R$ is the coverage line corresponding to $r_i \in R$. Since p_i is a straight line covering n_i grid cells (i.e., connecting the centers of the end grid cells in a row), the coverage line will have the length

$$L(p_i) = (n_i - 1)l. \tag{1}$$

Let P^* be the optimal coverage path for $\widetilde{\mathcal{W}}$ of length $L(P^*)$. To lower-bound $L(P^*)$, consider the length of a coverage path that covers every cell without any overlap (e.g., cells arranged in a loop). This gives

$$L(P^*) \geq \frac{\text{Area}}{l} = \frac{nl^2}{l} = nl. \tag{2}$$

We now derive an approximation factor for the rank-touring framework. Our derivation assumes access to an approximation algorithm to solve the metric traveling salesman problem (metric-TSP), where the edge costs of the graph satisfy the triangle inequality. Let γ be the approximation factor of this TSP solver.

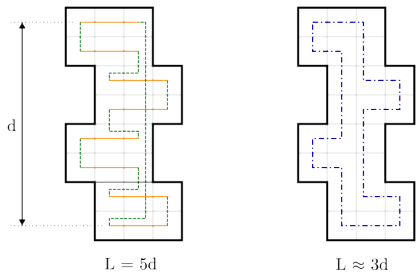


Fig. 1: An IOP where a rank-touring algorithm (left) computes a path of length $5/3$ times the optimal coverage path length (right).

Theorem 1. *Given an IOP $\widetilde{\mathcal{W}}$, the rank-touring framework computes a coverage path P of length $L(P)$ satisfying*

$$L(P) \leq (1 + 1.5\gamma)L(P^*),$$

where P^* is the optimal coverage path of $\widetilde{\mathcal{W}}$.

Proof. Let $R = \{r_1, \ldots, r_m\}$ be a rank partition of $\widetilde{\mathcal{W}}$, and let $P_R = \{p_1, \ldots, p_m\}$ be the resulting coverage line set. The rank-touring framework computes a coverage path P by connecting the coverage lines in P_R using a set of transition paths P_T. The resulting length of P is

$$L(P) = L(P_R) + L(P_T) = \sum_{i=1}^{m} L(p_i) + L(P_T).$$

We bound $L(P)$ by bounding the above sum of path lengths. Following Eqs. (1) and (2), we have

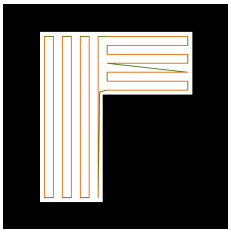
$$\sum_{i=1}^{m} L(p_i) = (n - m)l \leq nl \leq L(P^*).$$

Now, consider a tour T_e of length $L(T_e)$ over the *endpoints* of P_R computed by the γ-approximation metric-TSP solver using collision-free paths in $\widetilde{\mathcal{W}}$. Note that T_e does not necessarily connect the coverage line set P_R to form a coverage path. Let T_e^* be the optimal TSP tour of these endpoints. Then, we have

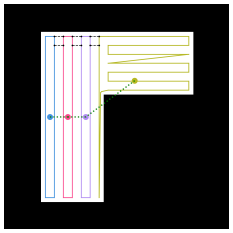
$$L(T_e) \leq \gamma L(T_e^*) \leq \gamma L(P^*).$$

The second inequality follows as T_e^* covers fewer grid cells than P^* (covers all cells). Consider the weighted graph $\bar{G}$ induced by T_e, where the edge costs are the path lengths between endpoints. To this, we add edges corresponding to the coverage lines P_R. The resulting $\bar{G}$ has a total edge cost $\leq (1 + \gamma)L(P^*)$ and all of its vertices have odd degree. However, the edges of T_e form two perfect matchings of the endpoints of P_R. Let M^* be the matching with cost $\leq L(T_e)/2$. Adding the edges of M^* to $\bar{G}$ gives a graph with all even degree vertices containing an Eulerian cycle of cost $\leq (1 + 1.5\gamma)L(P^*)$. This proves the theorem, as this cycle's cost is a bound on $L(P)$. □

Obtaining a tight example for this bound is non-trivial without considering environments with unrealistic perturbations.



(a) Coverage ranks (orange) with matching edges (green)



(b) Sub-tour merging with spanning tree

Fig. 2: MLC-Approx stages: (a) obtain coverage lines (orange) from the minimum rank partition and compute min-length perfect matching paths (green), and (b) identify the resulting sub-tours (colored paths) and evaluate possible merges (black lines) as determined by an ordering (dotted green lines).

However, we provide an algorithm for which this factor must be at least $5/3$.

Proposition 1. *There exists an IOP $\widetilde{\mathcal{W}}$ for which a rank-touring algorithm computes a coverage path of length $5/3$ times the optimal coverage path length.*

Proof. Consider the IOP in Fig. 1, where d is its height measured between the center points of the top and bottom grid cells. We can stack copies of this IOP vertically to obtain a new IOP with an arbitrarily large d. Now, consider a rank-touring algorithm that (i) partitions the IOP into the minimum number of *horizontal* ranks, and (ii) computes a minimum-length tour connecting the coverage lines. The resulting coverage path, shown in Fig. 1 (left), has length $5d$ for a large d. However, the IOP allows a Hamiltonian cycle (right of Fig. 1) with length $3d + 3l \approx 3d$, i.e., the optimal coverage path. Therefore, the algorithm computes a coverage path of length $5/3$ times the optimal. □

IV. A $(1.5 + \epsilon)$-Approximation Algorithm

We now present a $(1.5 + \epsilon)$-approximation algorithm for Problem 2, called the Minimum Length Coverage Approx (MLC-Approx) algorithm. Instead of connecting ranks at their endpoints, MLC-Approx allows the breaking and merging of neighboring ranks while minimizing the path length added to the final coverage path. We consider that the IOP is *connected*, i.e., there exists a path between every pair of grid cells with *axis-parallel* motions.

A. General Framework

A pseudocode of the coverage planning approach is given in Algorithm 1. We start by computing the *minimum* rank partition R of the IOP $\widetilde{\mathcal{W}}$ and obtain the resulting coverage lines P_R for the IOP. For this, we use the approach proposed in [8], which utilizes a linear program (LP) to compute the optimal partition in polynomial time. Fig. 2(a) shows the resulting set of coverage lines for an example IOP.

Let V be the set of all the rank endpoints, where each $v \in V$ corresponds to the center of a grid cell at one end of a rank. We compute a *perfect matching* of V that minimizes the length of the paths connecting the endpoints in the matching (Line 1). For this, we use a transition graph $G = (V, E)$ where the edges E represent collision-free *transition paths* between two endpoints with the edge weights being the lengths of the transition paths. The result is a set M of matching paths connecting pairs of rank endpoints. Together, the matching paths M and the coverage lines P_R form a set $\mathcal{T}$ of *sub-tours*, i.e., closed paths that collectively cover all cells in the IOP, where $|\mathcal{T}| \geq 1$ (Line 2). Fig. 2 shows an example set of sub-tours, where the coverage lines (orange) and matching paths (green) together form a set of sub-tours (colored paths in Fig. 2(b)) covering the IOP.

Algorithm 1 MLC-Approx

Input: Environment IOP $\widetilde{\mathcal{W}}$, Coverage lines P_R from rank partition, Transition Graph $G = (V, E)$ of rank endpoints
Output: Coverage Path P

1: $M \leftarrow$ A min-length perfect matching of V using G
2: $\mathcal{T} \leftarrow$ Sub-tours obtained from combining P_R and M
3: $S_{\mathcal{T}} \leftarrow$ SubtourOrdering$(\mathcal{T})$ ▷ Ordering of adjacent sub-tour pairs to be merged
4: **for** $(S_i, S_j) \in S_{\mathcal{T}}$ **do**
5: Merge S_i with S_j ▷ See Section IV-B
6: $P \leftarrow$ Final tour after merging all sub-tours
7: **return** P

We now look to merge the sub-tours in $\mathcal{T}$ into a single coverage path P. For this, we use the connectivity of the IOP and the resulting adjacency of the sub-tours.

Definition 1 (Adjacent Sub-tours)**.** *Two sub-tours are adjacent if they each cover grid cells that are axis-parallel (horizontal/vertical) neighbors of each other.*

Since the IOP is connected, every sub-tour in $\mathcal{T}$ is adjacent to at least one other sub-tour. Therefore, we use a sub-routine named SubtourOrdering$(\mathcal{T})$ to determine an ordering of adjacent sub-tour pairs that can be merged to obtain P (Line 3). In our implementation of SubtourOrdering$(\mathcal{T})$, we compute a *spanning tree* in an adjacency graph $G' = (V', E')$, where each sub-tour is represented by a vertex $v \in V'$ and edges $e \in E'$ connect two adjacent sub-tours. Fig. 2(b) shows the spanning tree of sub-tours in an example environment. We use the edges of this spanning tree to determine pairs of adjacent sub-tours that can be merged in any order to obtain P (Lines 4-6). In the next sub-section, we address how two sub-tours are merged in MLC-Approx.

B. Sub-tour Merging

We propose two strategies to merge adjacent sub-tours, namely *double edge* merging and *corner junction* merging. In both strategies, we exploit the layout of the adjacent (neighboring) grid cells covered by two sub-tours to determine a merging strategy that limits the path length added. For the rest of this section, let the two sub-tours to be merged be represented by S and $\bar{S}$. With some abuse of notation, we say that a grid cell $c_i \in S$ if sub-tour S covers the cell c_i.

Double Edge Merging: A simple strategy to merge sub-tours is to add a *double edge* between the two adjacent grid

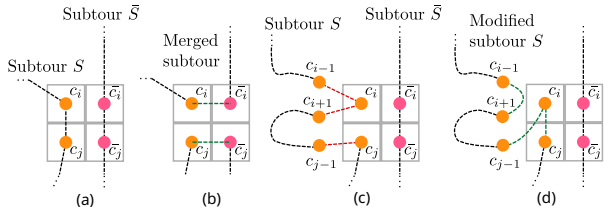


Fig. 3: Corner junction merging: (a) Corner junction formed by 4 grid cells. (b) Edge swap to merge two sub-tours at the junction. (c & d) Modifying sub-tour S to allow an edge swap by removing the red edges and adding the green edges.

cells $c_i \in S$ and $\bar{c}_i \in \bar{S}$. Specifically, we add a traversal path from c_i to $\bar{c}_i$ and another path from $\bar{c}_i$ to c_i. During coverage, the coverage tool follows this double edge, where it moves from sub-tour S to sub-tour $\bar{S}$, covers $\bar{S}$, and returns to finish covering S. Given that the coverage tool width is l, the length added to the coverage path using this strategy is $2l$. Additionally, since we consider the IOP to be connected, all sub-tours arising from the perfect matching can be merged using double edges to form a single tour.

Corner Junction Merging: We now consider a strategy to reduce the length added during the merge. Consider the case where two adjacent grid cells $c_i, c_j \in S$ *share a corner* with two adjacent grid cells $\bar{c}_i, \bar{c}_j \in \bar{S}$. We refer to these four grid cells as a *corner junction* (Fig. 3). Now, we evaluate a set of possible merges within this junction that limit the path length added. First, consider the case where sub-tour S has an edge (c_i, c_j) and sub-tour $\bar{S}$ has an adjacent and parallel edge $(\bar{c}_i, \bar{c}_j)$ connecting the corresponding pairs of cells (Fig. 3(a)). In this case, we simply perform an *edge swap*, where we replace these edges with new ones $(c_i, \bar{c}_i)$ and $(c_j, \bar{c}_j)$ that merge the sub-tours (Fig. 3(b)). This swap adds no path length to the resulting merged sub-tour.

Now, consider the case where sub-tour S has no edge (c_i, c_j) parallel to $(\bar{c}_i, \bar{c}_j)$. Instead, let c_{i-1}, c_{i+1} be the cells covered by sub-tour S before and after c_i respectively. Also, let c_{j-1} be the cell covered by sub-tour S before c_j. Fig. 3(c) shows an example junction. For now, assume that an edge $(\bar{c}_i, \bar{c}_j)$ exists in sub-tour $\bar{S}$ (we will remove this assumption later). To merge the sub-tours, we first modify sub-tour S by removing three edges, namely $(c_{i-1}, c_i), (c_i, c_{i+1})$, and (c_{j-1}, c_j), and replacing them with three new edges $(c_{i-1}, c_{i+1}), (c_{j-1}, c_i)$, and (c_i, c_j) (Fig. 3(d)). The path resulting from these edge swaps is still a closed sub-tour, but more importantly, there is now an edge connecting c_i and c_j in the sub-tour. We now merge the sub-tours using the edge swap from before. For edge (c_{j-1}, c_i), let $L(c_{j-1}, c_i)$ be the length of the edge (the shortest obstacle-free path between the cells). WLOG, we assume that $L(c_{j-1}, c_i) \leq L(c_{i+1}, c_j)$. Specifically, between two choices for the above sub-tour modification, we chose the one with the least added length. While this added length can be computed after the sub-tour modification is made, we speed up this process by obtaining a bound on the added length that is easier to compute.

Proposition 2. *Given two sub-tours S and $\bar{S}$ with a corner junction as defined above, merging the sub-tours at the corner junction when there is no edge (c_i, c_j) results in added length of at most $L(c_{j-1}, c_i)$.*

Proof. The length of the edges added by the modification is:

$$L^+ = L(c_{i-1}, c_{i+1}) + L(c_{j-1}, c_i) + L(c_i, c_j).$$

Similarly, the length removed by the modification is:

$$L^- = L(c_{i-1}, c_i) + L(c_i, c_{i+1}) + L(c_{j-1}, c_j).$$

However, following the triangle inequality, we also have:

$$L(c_{i-1}, c_{i+1}) \leq L(c_{i-1}, c_i) + L(c_i, c_{i+1}).$$

Subtracting L^- from L^+, we have the added path length:

$$\begin{aligned} L^+ - L^- = \quad & L(c_{i-1}, c_{i+1}) + L(c_{j-1}, c_i) + L(c_i, c_j) - \\ & L(c_{i-1}, c_i) - L(c_i, c_{i+1}) - L(c_{j-1}, c_j)). \end{aligned}$$

Following the inequality and that $L(c_{j-1}, c_j) \geq L(c_i, c_j) = l$, we can rewrite the above as:

$$L^+ - L^- \leq L(c_{j-1}, c_i),$$

which proves the proposition. □

Following this, we identify a sub-tour modification that minimizes the added length by checking the length of the *"cross path"* (i.e., the corresponding $L(c_{j-1}, c_i)$) added. If $L(c_{j-1}, c_i) > 2l$, we use the double edge strategy for the merge using one of two grid cell pairs in the corner junction. For the case when sub-tour $\bar{S}$ also does not have an edge $(\bar{c}_i, \bar{c}_j)$ between its adjacent cells, we modify $\bar{S}$ as above to create an edge at the junction. We check if the *total* length added by modifying both subtours is greater than $2l$, at which point we simply defer to the double edge strategy.

The above strategies enable the merging of two sub-tours at any set of adjacent cells or corner junctions, while ensuring that the path length added by each merge does not exceed $2l$. In Line 5 of Algorithm 1, we choose the merge with the least (potentially zero) added path length.

C. Approximation on Path Length

We now show that the above approach yields the following approximation factor on the resulting coverage path length.

Theorem 2. *Given a connected IOP with n grid cells, and a rank partition R containing m ranks, the coverage path P obtained using MLC-Approx has length $L(P)$ satisfying*

$$L(P) \leq (1.5 + m/n)L(P^*),$$

where P^* is the optimal coverage path of the IOP.

Proof. Let $L(P_R)$ be the total length of the coverage lines P_R obtained from the ranks R. Recall that

$$L(P_R) = (n - m)l.$$

Let $L(M)$ be the total length of the matching paths M computed by MLC-Approx to form the sub-tours. Now, let T_V^* be the optimal TSP tour of the rank endpoints V. Since M is a minimum-length perfect matching of V, we have

$$L(M) \leq 0.5L(T_V^*) \leq 0.5L(P^*).$$

Now, we consider the length added to the path from merging adjacent sub-tours. We have the number of sub-tours to be at most m, i.e., the number of ranks. Since we limit the length added from each merge to $2l$ (double edge strategy), the total path length added is $2ml$. Putting it all together, we have the following for the total path length:

$$\begin{aligned} L(P) &= L(P_R) + L(M) + 2ml \\ &\leq (n-m)l + 0.5L(P^*) + 2ml \\ &= nl + 0.5L(P^*) + ml = (1 + m/n)nl + 0.5L(P^*). \end{aligned}$$

Following that $L(P^*) \geq nl$ (Eq. 2), we obtain

$$L(P) \leq (1.5 + m/n)L(P^*),$$

which proves the theorem. □

The factor in Theorem 2 has an additive m/n term. This term has little effect as (i) using a minimum rank partition (minimizing m), we observe that $m/n \ll 1$ is small, and (ii) many of the sub-tour merges can be done with the corner junction strategy, which adds little to no path length. With this, we see that the approximation guarantee is closer to 1.5.

D. Turn Minimization

While the use of a minimum rank partition minimizes the number of ranks in the coverage path, we must also minimize the turns added during the sub-tour merges. To do this, we use a *weighted* sub-tour adjacency graph to compute the ordering in Line 3 of MLC-Approx (Algorithm 1), where the edge weights are given by the minimum number of turns added by merging two adjacent sub-tours. We then compute a *minimum* spanning tree (MST) in this graph to determine the sub-tour pairs to be merged. When merging two adjacent sub-tours (as per MST edge), we pick the merge strategy that minimizes the number of added turns. Typically, this leads to merges occurring at the ends of ranks, as they add fewer additional turns to the final coverage path.

V. MLC-Approx with Lazy Evaluations

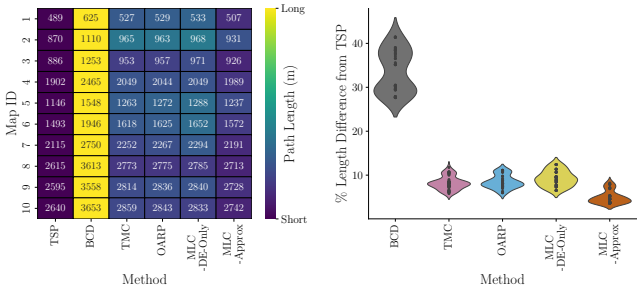
For large maps, computing a fully-connected transition graph $G = (V, E)$ to perform perfect matching of all rank endpoints can be computationally expensive, as it requires planning collision-free paths between all combinations of endpoints ($O(V^2)$ operations). However, we exploit the use of perfect matching to employ a *lazy* strategy, where we do not require exploring all the paths in the transition graph.

We start with a fully-connected transition graph $\tilde{G} = (V, \tilde{E})$, where $\tilde{E}$ is the set of *Euclidean* edges (may have collisions) between each pair of endpoints in V. We then replace Line 1 in MLC-Approx with the following. We iteratively compute a matching $\tilde{M}$ using $\tilde{G}$, where $\tilde{M}$ may contain both Euclidean and collision-free edges. For all Euclidean edges in $\tilde{M}$, the corresponding edge in $\tilde{E}$ is replaced with a collision-free edge, and the edge cost is updated accordingly. This process is repeated until $\tilde{M}$ has no Euclidean edges, i.e., $\tilde{M} = M$. While the lazy approach may still require $O(V^2)$ edge replacements, we observe that this number is far less in practice. Since the length of the Euclidean edge is a lower bound on the length of the collision-free path, the final matching M is optimal.

For some complex environments, one may additionally use a *difference threshold* heuristic to reduce the number of matching iterations. If the length difference between a Euclidean edge and its replacement exceeds this threshold, we preemptively replace all other incident edges. This still only replaces a fraction of the edges while reducing the number of iterations, improving runtime.

TABLE I: Statistics for dataset maps.

Statistic	Min	Max	Mean	Std Dev
Grid Cells	610	3262	2079	953
Minimum Number of Ranks	39	290	147	73
TSP Path Length (m)	489	2640	1675	776



(a) (b)

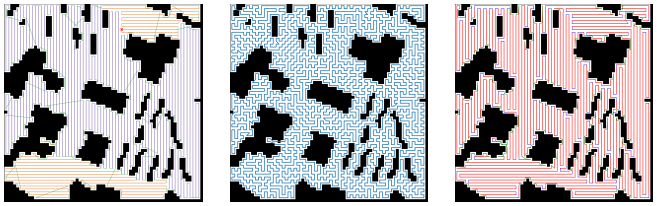
Fig. 4: Path length comparisons. (a) Average path lengths for each map. (b) % length difference from TSP across all maps.

VI. Simulation Results

While the approximation factor is the focus of this paper, we explore how MLC-Approx performs in practice compared to state-of-the-art approaches. We conduct simulations using a dataset of 10 real-world environments for which the statistics are given in Table I. The dataset, which is available at the link https://github.com/megalphian/MLC-Maps, includes a combination of street maps from [21] and anonymized maps of real-world environments provided by our industry partner, Avidbots. To implement MLC-Approx, we obtain the minimum number of axis-parallel ranks for each map's IOP by solving the LP from [8]. The map dataset is sorted in increasing order of complexity as determined by the number of ranks computed by the LP for each map. We compute the minimum-length perfect matching using the lazy approach (Section V). We then use the strategies from Sections IV-B and IV-D to merge the resulting sub-tours into a coverage path while minimizing turns added to the path.

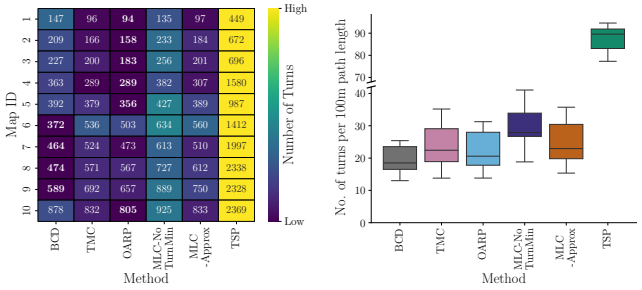
A. Path Length Comparisons

We compare MLC-Approx against three baseline approaches: (i) the OARP approach from [8], (ii) the turn-minimizing coverage (TMC) approach from [7], and (iii) the boustrophedon cell decomposition (BCD) approach from [11]. OARP and TMC are turn-minimizing CPP approaches that use the rank-touring framework. TMC computes ranks using an



(a) (b) (c)

Fig. 5: Coverage paths for an example environment generated using (a) OARP, (b) TSP, and (c) the proposed MLC-Approx.



(a) (b)

Fig. 6: Turn comparisons. (a) Number of turns (bold - fewest turns). (b) Coverage path straightness as denoted by the number of turns per 100 meters of path length.

iterative heuristic that can be run for a long time to find high-quality solutions. To standardize the comparison, the runtime of this heuristic is limited to that of OARP's rank partitioning step. All approaches are tested on each dataset map for 5 trials. For each map, the grid-based approaches (MLC-Approx, OARP, and TMC) must cover the same grid approximation as determined by a tool width $l = 0.8$ m (to match our industry partner's robot). For OARP and TMC, a GTSP solver [22] is used to compute a minimum-length tour connecting the ranks, where the solver is given a fully-connected transition graph. In addition to these baselines, we also perform an ablation analysis by comparing the full MLC-Approx approach with a variant that only uses double edge (DE) merging. We will refer to this as MLC-DE-Only.

To compare against an optimal planner, we compute a coverage path for each map by solving the Euclidean Travelling Salesman Problem (TSP) of the grid cells. The TSP path will always have the shortest length (does not minimize turns), so the length difference to TSP gives the gap to the optimal path length. The LKH3 solver [23] was used to solve TSP with the Euclidean distance between grid cells as the edge costs (may have collisions). Fig. 4(a) reports the average path lengths across trials for each approach and map, and Fig. 4(b) compares the *% length difference* from TSP for all approaches. MLC-Approx outperforms all benchmarks and comes closest to the TSP length. We attribute this to the sub-tour merging strategies (Section IV-B) that re-use path length instead of adding length to connect ranks. We see this effect in Fig. 5, where the MLC-Approx path has fewer overlapping transition paths (green lines) than the OARP path, resulting in an overall shorter path. MLC-Approx also outperforms BCD, mainly because BCD paths have many overlapping sections as it aims to cover 100% of the environment. However, we observe that grid-based approaches achieve an average coverage of 96% of the environments in our dataset. So, while MLC-Approx does not cover the entire environment, it covers most of the reachable regions.

MLC-Approx path lengths are also closer to TSP than MLC-DE-Only, supporting our earlier claim that most sub-tours are merged with corner junction merging, which adds little to no length. As such, MLC-Approx path lengths are closer to the optimal than the $(1.5 + m/n)L(P^*)$ bound.

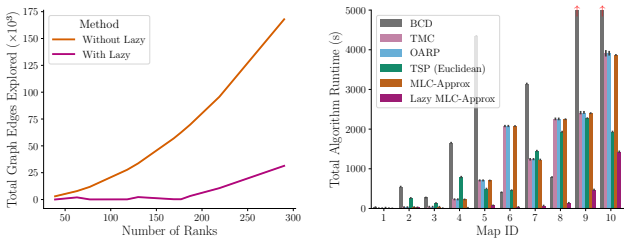
B. Turns Comparisons

We now compare the number of turns in the coverage paths generated by MLC-Approx. For the ablation analysis, we also include a variant of MLC-Approx without turn minimization (Section IV-D), which we refer to as MLC-NoTurnMin. Fig. 6(a) shows the number of turns for each CPP approach to cover each map, averaged across trials. While MLC-Approx has fewer turns compared to MLC-NoTurnMin, MLC-Approx has more turns than the other turn-minimizing approaches. This is because sub-tour merges often add turns to the path as a result of breaking ranks to re-use path length. We analyze this tradeoff between length and turns by comparing the number of turns per 100 meters of path length, i.e., a measure of path "straightness". Fig. 6(b) shows this comparison, where MLC-Approx paths have comparable straightness to the turn-minimizing baselines. Compared to TSP, MLC-Approx paths have significantly fewer turns despite having comparable path lengths. While this comparison is not entirely fair, we use it to show that MLC-Approx approaches TSP-like path lengths with comparable straightness (turns/length tradeoff) to that of turn-minimizing approaches. While one can also compare against angular-metric TSP [24], we found that, while it provides good solutions for smaller maps of ~ 100 grid cells, it scales poorly for the large maps in our dataset.

BCD achieves the fewest turns for some maps due to the presence of non-orthogonal boundaries in these maps (e.g., walls oriented at different angles). Unlike grid-based approaches, BCD is not limited to covering in axis-parallel directions, and so its paths align better with the environment. A future improvement of MLC-Approx is allowing multiple coverage directions while retaining its guarantees.

C. Runtime Improvements with Lazy Evaluations

We demonstrate the runtime improvements from using MLC-Approx with lazy evaluations. For the lazy method, we used a difference threshold of 8 meters (10 times l) for all maps to standardize the comparison. Since perfect matching can typically be computed efficiently, the runtime bottleneck for MLC-Approx is the number of edge *evaluations*, i.e., computing collision-free paths connecting ranks. We plan these paths using a visibility graph planner [25]. At worst, both lazy MLC-Approx and non-lazy approaches [7], [8] evaluate all $O(V^2)$ edges in the transition graph. However, we plot the number of edge evaluations for lazy and non-lazy MLC-Approx in Fig. 7(a), where we observe that lazy MLC-Approx



(a) (b)

Fig. 7: Runtime comparisons. (a) Number of edge evaluations (i.e., obstacle-free paths planned) by MLC-Approx with/without lazy. (b) Runtimes for each map (red arrow - exceeds 5000s).

evaluates a small fraction of all edges. This results in a runtime performance boost compared to non-lazy approaches. Fig. 7(b) shows the results of comparing the runtimes of each approach, where lazy MLC-Approx achieves the fastest runtimes among all baselines. As discussed in Section V, lazy MLC-Approx computes an optimal matching of rank endpoints, so it does not sacrifice the path quality.

Here, the TSP planner appears to be faster than non-lazy approaches while having more edges to evaluate. This is because we used Euclidean distances for the TSP costs, which are quicker to compute than visibility graph paths. Using a visibility planner, the runtime to compute a complete graph for TSP rises to a prohibitive amount (multiple hours).

VII. Conclusion

We studied minimum-length coverage path planning (CPP) for robots using grid decompositions of environments. The rank-touring framework for CPP [7]–[9] guarantees a coverage path of length $(1+1.5\gamma)$ times the optimal, where $\gamma \geq 1$ is the approximation factor for metric-TSP. To improve this bound, we proposed MLC-Approx, which modifies rank-touring to achieve an approximation factor of $(1.5+\epsilon)$, where $\epsilon \ll 1$ depends on the number of ranks (straight-line paths). In practice, MLC-Approx plans paths with near-optimal lengths compared to state-of-the-art approaches [7], [8], [11]. MLC-Approx paths also have comparable "straightness" to turn-minimizing CPP planners. We proposed a lazy variation of MLC-Approx for efficient computation of these paths.

The core techniques of MLC-Approx (matching and sub-tour merging) are applicable to any rank-touring approach that minimizes ranks. For real-world deployment, MLC-Approx paths should be tracked using a local planner similar to the setup from [8]. Since this is a rank-based approach, coverage is mostly performed along straight-line paths, which robots can track with high accuracy. The local planner can also allow replans to avoid small unknown obstacles in the environment. For large obstacles, MLC-Approx with lazy evaluations allows for fast replanning of coverage paths. Future improvements to MLC-Approx include adding a configurable length/turns trade-off and using multi-directional environment decompositions for CPP [6], [14].

References

[1] J. Palacios-Gasós, Z. Talebpour, E. Montijano, C. Sagüés, and A. Martinoli, "Optimal path planning and coverage control for multi-robot persistent coverage in environments with obstacles," in *IEEE Intl. Conf. on Robotics and Automation*, May 2017, pp. 1321–1327.

[2] X. Jin and A. Ray, "Coverage control of autonomous vehicles for oil spill cleaning in dynamic and uncertain environments," in *American Control Conference (ACC)*, 2013, pp. 2594–2599.

[3] K. Shah, A. Schmidt, G. Ballard, and M. Schwager, "Large Scale Aerial Multi-Robot Coverage Path Planning," *Field Robotics*, vol. 2, pp. 1971–1998, Mar. 2022.

[4] E. Arkin, S. Fekete, and J. S. Mitchell, "Approximation algorithms for lawn mowing and milling," *Computational Geometry*, vol. 17, no. 1-2, pp. 25–50, 2000.

[5] H. Choset and P. Pignon, "Coverage Path Planning: The Boustrophedon Cellular Decomposition," *Field and Service Robotics*, pp. 203–209, 1998.

[6] S. Agarwal and S. Akella, "Area coverage with multiple capacity-constrained robots," *IEEE Robotics and Automation Letters*, vol. 7, no. 2, pp. 3734–3741, Apr. 2022.

[7] I. Vandermeulen, R. Groß, and A. Kolling, "Turn-minimizing multirobot coverage," in *IEEE International Conference on Robotics and Automation (ICRA)*, 2019, pp. 1014–1020.

[8] M. Ramesh, F. Imeson, B. Fidan, and S. L. Smith, "Optimal Partitioning of Non-Convex Environments for Minimum Turn Coverage Planning," *IEEE Robotics and Automation Letters*, vol. 7, no. 4, pp. 9731–9738, Oct. 2022.

[9] E. M. Arkin, M. A. Bender, E. D. Demaine, S. P. Fekete, J. S. B. Mitchell, and S. Sethia, "Optimal Covering Tours with Turn Costs," *SIAM Journal on Computing*, vol. 35, no. 3, pp. 531–566, 2005.

[10] S. P. Fekete and D. Krupke, "What Goes Around Comes Around: Covering Tours and Cycle Covers with Turn Costs," *Theory of Computing Systems*, vol. 68, no. 5, pp. 1207–1238, Oct. 2024.

[11] R. Bähnemann, N. Lawrance, J. J. Chung, M. Pantic, R. Siegwart, and J. Nieto, "Revisiting Boustrophedon coverage path planning as a generalized traveling salesman problem," in *Field and Service Robotics*, 2021, pp. 277–290.

[12] E. Galceran and M. Carreras, "A survey on coverage path planning for robotics," *Robotics and Autonomous Systems*, vol. 61, no. 12, pp. 1258–1276, 2013.

[13] R. Bormann, F. Jordan, J. Hampp, and M. Hägele, "Indoor Coverage Path Planning: Survey, Implementation, Analysis," in *IEEE International Conference on Robotics and Automation*, 2018, pp. 1718–1725.

[14] W. Huang, "Optimal line-sweep-based decompositions for coverage algorithms," in *IEEE Int Conf Robotics and Autom*, 2001, pp. 27–32.

[15] S. Bochkarev and S. L. Smith, "On minimizing turns in robot coverage path planning," in *2016 IEEE International Conference on Automation Science and Engineering (CASE)*, 2016, pp. 1237–1242.

[16] Y. Gabriely and E. Rimon, "Spanning-tree based coverage of continuous areas by a mobile robot," *Annals of Mathematics and Artificial Intelligence*, vol. 31, no. 1-4, pp. 77–98, 2001.

[17] J. Lu, B. Zeng, J. Tang, T. Lam, and J. Wen, "TMSTC*: A path planning algorithm for minimizing turns in multi-robot coverage," *IEEE Robotics and Automation Letters*, vol. 8, no. 8, pp. 5275–5282, 2023.

[18] Y. Gabriely and E. Rimon, "Competitive on-line coverage of grid environments by a mobile robot," *Computational Geometry*, vol. 24, no. 3, pp. 197–224, 2003.

[19] J. Song and S. Gupta, "ϵ^*: An Online Coverage Path Planning Algorithm," *IEEE Transactions on Robotics*, vol. 34, no. 2, pp. 526–533, 2018.

[20] X. Kan, H. Teng, and K. Karydis, "Online Exploration and Coverage Planning in Unknown Obstacle-Cluttered Environments," *IEEE Robotics and Automation Letters*, vol. 5, no. 4, pp. 5969–5976, 2020.

[21] N. R. Sturtevant, "Benchmarks for Grid-Based Pathfinding," *IEEE Transactions on Computational Intelligence and AI in Games*, vol. 4, no. 2, pp. 144–148, 2012.

[22] S. L. Smith and F. Imeson, "GLNS: An effective large neighborhood search heuristic for the Generalized Traveling Salesman Problem," *Computers & Operations Research*, vol. 87, pp. 1–19, 2017.

[23] K. Helsgaun, "An extension of the lin-kernighan-helsgaun tsp solver for constrained traveling salesman and vehicle routing problems," *Roskilde: Roskilde University*, vol. 12, pp. 966–980, 2017.

[24] A. Aggarwal, D. Coppersmith, S. Khanna, R. Motwani, and B. Schieber, "The Angular-Metric Traveling Salesman Problem," *SIAM Journal on Computing*, vol. 29, no. 3, pp. 697–711, Jan. 2000.

[25] K. J. Obermeyer and Contributors, "VisiLibity: A C++ Library for Visibility Computations in Planar Polygonal Environments," http://www.VisiLibity.org, 2008.