Optimal Partitioning of Non-Convex Environments for Minimum Turn Coverage Planning

Megnath Ramesh, Frank Imeson, Baris Fidan, and Stephen L. Smith

Abstract—In this paper, we tackle the problem of planning an optimal coverage path for a robot operating indoors. Many existing approaches attempt to discourage turns in the path by covering the environment along the least number of coverage lines, i.e., straight-line paths. This is because turning not only slows down the robot but also negatively affects the quality of coverage, e.g., tools like cameras and cleaning attachments commonly have poor performance around turns. The problem of minimizing coverage lines however is typically solved using heuristics that do not guarantee optimality. In this work, we propose a turn-minimizing coverage planning method that computes the optimal number of axis-parallel (horizontal/vertical) coverage lines for the environment in polynomial time. We do this by formulating a linear program (LP) that optimally partitions the environment into axis-parallel ranks (non-intersecting rectangles of width equal to the tool width). We then generate coverage paths for a set of real-world indoor environments and compare the results with state-of-the-art coverage approaches.

I. INTRODUCTION

Coverage path planning is an automation challenge in which a robot must find an optimal path such that its tool or sensor covers the entire environment [1]. This problem has a wide range of applications, including cleaning [2], agriculture [3], visual inspection [4]–[6], and more recently, autonomous disinfection of hospitals during the COVID-19 pandemic [7]. The primary objective in coverage planning is to minimize overlap or "double" coverage. This is typically achieved through a lawnmower-style path consisting of parallel non-overlapping coverage lines. Thus, the main differentiation between coverage plans is the transitions between these lines, where the robot performs little to no additional coverage. This has resulted in a growing body of work focused on *minimizing turns*, i.e., minimizing the time spent transitioning between coverage lines. Turns also have the following adverse effects: (i) the robot travels slower around turns which increases total coverage time, (ii) in cleaning applications, the tool does not properly pick up water and dust while turning, and (iii) in sample retrieval applications, the robots may experience high pose estimation errors and poor sensor coverage quality during turns [8].

Coverage planning with minimum turns is however an NP-Hard problem [9]. As such, a common solution framework is to simplify the problem into three steps:



Fig. 1: (a) An example environment and (b) its optimal rank partitioning obtained using the proposed OARP method: the orange partitions are horizontal ranks and the purple are vertical ranks.

- (i) decompose the environment into sub-regions or cells,
- (ii) use the cells to help *place coverage lines* (straight-line paths) in the environment, and
- (iii) construct the coverage path by computing a *tour* of the coverage lines.

For example, exact decomposition methods like [10] decompose the environments into convex cells, each of which is covered by a lawnmower-style path. Following this framework, the robot only turns when transitioning from one coverage line to another. As a result, the number of turns in the path is often reduced by minimizing the number of coverage lines.

A state-of-the-art method which follows this general framework is Vandermeulen et. al. [11], where they aim to cover the environment along a minimum number of axisparallel (horizontal/vertical) coverage lines. Such a method is particularly well suited for indoor environments, in which many walls are either parallel or orthogonal. Their method decomposes the environment into cells, which are then used to partition the environment into *ranks*, i.e., thin disjoint rectangles of width equal to the robot's tool width, where the center line of each rank along its length defines a coverage line. However, to solve the partitioning step, the authors propose an iterative heuristic with no optimality guarantees. In this paper, we propose a coverage planner that follows the above framework and plans axis-parallel coverage paths with theoretical guarantees and with better performance.

Contributions: Our specific contributions are as follows:

- We prove that the axis-parallel rank partitioning problem is tractable and propose a polynomial-time solver that is guaranteed to find the optimal partition. We do this by posing the problem as a mixed integer linear program (MILP) and then proving that the linear relaxation provides optimal solutions.
- 2) We develop the Optimal Axis-Parallel Rank

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

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

Partitioning (OARP) method that leverages the above solver and plans a coverage path for non-convex environments.

3) We present experimental results using maps of realworld environments and show that OARP outperforms the state-of-the-art method from [11] in minimizing the decomposition time, number of ranks, number of turns, and consequently the cost of the coverage plan.

Related Work: Apart from [11], there are coverage planning methods in the literature that follow the same framework. Detailed surveys of such coverage planning methods can be found in [1] and [12]. One category of approaches is *exact decomposition*, where the environment is directly decomposed into (usually convex) cells without any prior approximation. Boustrophedon [10] is a widely used method of this category, where each cell is individually covered in a back-and-forth sweeping pattern along parallel coverage lines. There are also turn-minimizing variations of boustrophedon which aim to minimize the number of turns to cover each cell by determining a locally optimal coverage orientation for the cell. These methods use different criteria to determine a cell's orientation: by analysing the geometry of each cell [8], [13], [14] or its convex hull [15], or minimizing a cost function that models the robot's turns [16]. While these variations work well to incorporate multiple coverage directions to minimize turns, cell decomposition can be time-intensive for complex real-world environments. Also, these approaches do not guarantee that the total number of turns in the path is minimized. In this paper, we address these shortcomings by creating an optimal axis-parallel rank partition of the environment that implicitly minimizes the total number of turns without incorporating complex cell decomposition.

Another category of approaches, referred to as grid-based approaches, involve decomposing the environment into a set of uniform grid cells, where the robot must cover all accessible grid cells. Grid decomposition approximates the environment's boundary and obstacles and captures the areas coverable by the tool. Our proposed approach falls under this category. The shape of the grid cells is usually a square [17] but there are approaches that use hexagons [18]. Coverage planning for a grid environment can be solved by generating a spanning-tree of the grid cells [19] to find minimum length paths. A tour of the grid cells can also be obtained by solving the Travelling Salesman Problem (TSP) to generate a coverage plan [12]. Recently, learning-based coverage planning methods have also been introduced to solve this problem [20]-[23]. These approaches however do not minimize the number of turns while our proposed method does

Organization: In Section II, we define our decomposition approach and introduce the rank partitioning problem. In Section III, we formulate this problem as a Mixed Integer Linear Program (MILP) and then prove in Section IV that this problem can be relaxed to an LP and optimally solved in polynomial time. In Section V, we briefly discuss how we compute a tour of the ranks using a Generalized Travelling



Fig. 2: An Integral Orthogonal Polygon with light squares representing the polygon and grey areas representing the boundaries and holes. Each grid cell is as wide as the coverage tool.

Salesman Problem (GTSP) solver to obtain the full coverage plan. In Section VI, we present our experimental results on a set of real-world environments and compare the performance of our algorithm with the method proposed in [11].

II. PROBLEM DEFINITION

In this section, we provide a brief background on coverage planning and introduce a simple decomposition of the environment to tackle turn-minimization. Finally, we define the partitioning and touring problems which we will be solving in the rest of the paper.

A. Coverage Planning Problem

Consider a closed and bounded set $W \subseteq \mathbb{R}^2$ representing all points within the boundaries of our 2D indoor environment. Let $\mathcal{O} \subset W$ denote the set of obstacles or inaccessible points in W. The set $\widetilde{W} = W \setminus \mathcal{O}$ therefore contains all accessible points in the environment. The goal of the coverage problem is to plan a path so that the robot's tool covers all accessible points in the given environment. The feasibility of this problem however depends on the tool's footprint, which we represent as $\mathcal{A}(\omega) \subset \mathbb{R}^2$ relative to the robot's position $\omega \in \mathbb{R}^2$. Conventionally, $\mathcal{A}(\omega)$ is represented by a simple geometric shape such as a square of a fixed width l > 0 centered at ω [1], [9], [11], [19].

Assumption 1. The coverage tool is a square with width l.

The square tool may not be able to cover the environment entirely as there will likely be coverage gaps near the boundaries. Let $\widehat{W} \subseteq \widetilde{W}$ be the points of the environment that can be covered by the tool. The goal of the coverage problem is then to plan a path $P \subseteq \widetilde{W}$ such that

$$\bigcup_{\omega_i \in P} \mathcal{A}(\omega_i) = \widehat{\mathcal{W}}$$

B. IOP Decomposition

In this paper, we look at an extension of this problem where we minimize the number of turns in P. Following from Assumption 1, we decompose the environment \widehat{W} using a square grid and focus on paths where the tool moves only in axis-parallel directions during coverage (horizontal and vertical). We refer to this decomposition as an Integral Orthogonal Polygon (IOP), which is a set of square grid cells of size $l \times l$ [9]. Fig. 2 shows an example IOP generated for a simple non-convex environment with one hole. Computing the IOP representation of the environment constitutes the decomposition step of the OARP method.

C. Minimum Rank Partitioning Problem

We now consider a partition that will help quantify the number of potential turns in the final coverage plan. We look at partitioning the environment into regions that will each be covered with a straight-line path (no turns). A *rank* is the cumulative footprint of the coverage tool when traversing a straight-line path. For a square coverage tool of width l, a rank is a rectangular region of width l.

In an IOP, the robot can perform coverage by moving its square tool from one grid cell to another. A rank in an IOP is therefore a series of horizontally or vertically adjacent grid cells (e.g. Fig. 3). Since each grid cell is of the same size as the coverage tool, a rank can be covered along a single coverage line (straight-line path) with endpoints at the centers of grid cells. We can thus define the main problem of this paper as follows.

Problem 1 (Minimum Rank Partitioning Problem). *Given* an IOP representation of \widehat{W} , compute a partition of the IOP into the minimum number of axis-parallel (horizontal and vertical) ranks.

From the rank partition determined in Problem 1, we generate a coverage path for the environment by solving the following problem.

Problem 2 (Tour Generation Problem). *Given a set of axisparallel ranks (partitioning of an IOP), compute a tour of the ranks that minimizes the total cost of transitions between ranks.*

In a tour of the ranks, the coverage tool moves from one rank's endpoint to another along transition paths determined by the robot's dynamics.

III. MILP FORMULATION

In this section, we formulate Problem 1 as a mixed integer linear program (MILP), which involves a set of variables and linear constraints. Consider an IOP with n grid cells, each of which is represented by a variable c_i in $C = \{c_1, c_2, \ldots, c_n\}$. We start by introducing an operator orient $(c_i) \in \{H, V\}$ for each grid cell c_i to represent its coverage orientation (H for horizontal or V for vertical). We also introduce a series of variables that bound the number of ranks in the final partition. The goal of this MILP is to compute the orientations of grid cells that minimize the number of ranks.

Next, we use the concept of neighbouring grid cells to help represent each rank and construct the objective function to minimize the number of ranks. We do this by formulating constraints that can be interpreted as a series of merges for neighbouring cells. We call a grid cell *merge-able* with its neighbour if: (i) the cell and its horizontal neighbour are horizontally oriented, or (ii) the cell and its vertical neighbour are vertically oriented. A rank is obtained by merging a set of merge-able cells, and so the rank partition of the IOP depends on the assigned cell orientations. Fig. 3 illustrates the intuition of determining ranks from orientations.



Fig. 3: Illustration of merging oriented grid cells to form ranks and the corresponding coverage lines. The cell orientations are determined by the MILP from Section III, and the merge-able cells are merged.

We now work towards constructing a set of operators that count the ranks for a given set of oriented cells. The following lemma highlights a useful relationship for this task.

Lemma 1. In any minimal rank partition, each rank terminates at cells with less than two merge-able neighbours.

Proof. By definition, each grid cell in the IOP can have at most two merge-able neighbours across an axis. Let $R = (r_1, r_2, \ldots, r_k)$ be an ordered set of k similarly-oriented cells $r_i \in C$ representing a rank along an axis. By our definition of R, these cells are connected in a sequential order (r_1, r_2, \ldots, r_k) .

Consider one of the endpoints of R: r_1 . By definition, r_1 and r_2 are merge-able neighbours. Let r_1 have another neighbour r_t that is covered by a rank \tilde{R} . If r_t is a mergeable neighbour of r_1 , then R and \tilde{R} can be merged to a single rank, thereby reducing the number of ranks and contradicting the optimality of the partition. This would mean one of two things: (i) r_t does not exist, i.e., r_t is a border or an obstacle, or (ii) r_t is not a merge-able neighbour. Similarly, we can show that the other endpoint r_k has only one merge-able neighbour r_{k-1} , which completes the proof of Lemma 1.

Given the above lemma, we can calculate the number of ranks in an optimal partition by counting one of the two unique endpoints of each rank. For the rest of this paper, we only consider detecting and counting the *left* and *top* endpoints of horizontal and vertical ranks respectively. The choice of left and top endpoints is arbitrary and does not effect the solution approach. To identify and count the rank endpoints at the IOP border or holes, we add some artificial cells next to the border cells. These cells are not assigned an orientation, but simply indicate whether there is a rank that has an endpoint at the border. We will refer to these cells as *border identifiers*.

The following logical statements summarize how we will constrain our endpoint counting variables. For each grid cell c_i , denoting the left and top neighbors of c_i by $left(c_i)$ and $top(c_i)$ respectively, we define our endpoint operators as follows:

$$\operatorname{end}_{\mathrm{H}}(c_{i}) = \begin{cases} 1 & \text{if } \operatorname{orient}(c_{i}) = H \text{ and} \\ & \operatorname{orient}(\operatorname{left}(c_{i})) = V \\ 1 & \text{if } \operatorname{orient}(c_{i}) = H \text{ and } \operatorname{left}(c_{i}) \\ & \text{is a border identifier} \\ 0 & \text{otherwise} \end{cases}$$

$$\operatorname{end}_{V}(c_{i}) = \begin{cases} 1 & \text{if } \operatorname{orient}(c_{i}) = V \text{ and} \\ & \operatorname{orient}(\operatorname{top}(c_{i})) = H \\ 1 & \text{if } \operatorname{orient}(c_{i}) = V \text{ and } \operatorname{top}(c_{i}) \\ & \text{is } a \text{ border identifier} \\ 0 & \text{otherwise} \end{cases}$$

where $\operatorname{end}_{H}(c_i)$ and $\operatorname{end}_{V}(c_i)$ are binary operators determining if c_i is a horizontal or a vertical endpoint respectively. We now use these functions to formulate the following binary programming problem:

$$\min \quad \sum_{i=0}^{n} \operatorname{end}_{H}(c_{i}) + \sum_{i=0}^{n} \operatorname{end}_{V}(c_{i})$$
(1)

s.t. orient
$$(c_i) \in \{H, V\}, \forall i \in \{1, 2, ..., n\}.$$
 (2)

Next, we will convert the above binary programming problem to a MILP by replacing the operators with a set of new variables. Let us start with $\operatorname{end}_{\mathrm{H}}(c_i)$ and create the auxiliary variable y_h^i that bounds $\operatorname{end}_{\mathrm{H}}(c_i)$. The upper index (i) of y_h^i matches the lower index of c_i . We also introduce a set of binary variables to denote whether a cell is horizontally or vertically oriented. For the horizontal case, we introduce x_h^i which equals 1 if $\operatorname{orient}(c_i) = H$ and 0 otherwise. For the left neighbour, $c_l = \operatorname{left}(c_i)$, we observe the following:

$$x_{h}^{i} - x_{h}^{l} = \begin{cases} 1 & \text{if orient}(c_{i}) = H \text{ and} \\ & \text{orient}(\operatorname{left}(c_{i})) = V \\ 1 & \text{if orient}(c_{i}) = H \text{ and } \operatorname{left}(c_{i}) \text{ is} \\ & \text{a border identifier} \\ -1 & \text{if orient}(c_{i}) = V \text{ and} \\ & \text{orient}(\operatorname{left}(c_{i})) = H \\ 0 & \text{otherwise} \end{cases}$$

The above, along with a non-negativity constraint, gives the following encoding of y_h^i :

$$y_h^i \ge x_h^i - x_h^l$$

$$y_h^i \ge 0 ,$$

where $y_h^i \ge \text{end}_H(c_i)$. With these constraints, we observe that minimizing y_h^i gives us an optimal solution where $y_h^i = \text{end}_H(c_i)$. We now determine the values of y_h^i for all grid cells using the following system of vector inequalities:

$$\boldsymbol{y_h} \ge A_H \boldsymbol{x_h} \tag{3}$$

$$\boldsymbol{y_h} \ge \boldsymbol{0}, \tag{4}$$

where y_h and x_h are *n*-dimensional vectors composed of the variables y_h^i and x_h^i respectively. On further inspection, A_H in (3) is the *node-arc incidence (NAI) matrix* [24] of the directed graph G_H (see Fig. 4) composed of all horizontal path flows for an IOP grid cell from its left neighbour/border identifier. Similarly, we encode $\operatorname{end}_V(c)$ using the auxiliary variable vector y_v and the NAI matrix A_V of the directed graph G_V (Fig. 4).



Fig. 4: The directed graphs G_H and G_V for an example IOP.

Using the defined matrices and vectors, we write the following MILP to solve Problem 1:

$$\min \sum_{i=0}^{n} y_h^i + \sum_{i=0}^{n} y_v^i$$
(5)

s.t.
$$A_H \boldsymbol{x_h} - \boldsymbol{y_h} \le \boldsymbol{0}$$
 (6)

$$A_V \boldsymbol{x_v} - \boldsymbol{y_v} \le \boldsymbol{0} \tag{7}$$

$$x_h + x_v = 1 \tag{8}$$

$$\boldsymbol{x_h}, \boldsymbol{x_v} \in \{0, 1\}^n \quad \boldsymbol{y_h}, \boldsymbol{y_v} \ge \boldsymbol{0}, \tag{9}$$

where 1 is the column vector of ones and 0 is the vector of zeros. Eq. (8) ensures that each grid cell is assigned one of two orientations (either x_h^i or x_v^i equals 1). The objective in Eq. (5) gives us the number of ranks while x_h and x_v gives us the corresponding grid cell orientations. The solution of the MILP gives the minimum rank partition as determined by the grid cell orientations, where we obtain the ranks by merging all merge-able cells.

We also obtain an LP relaxation from this MILP using Eqs. (5)-(8) and replacing the binary variables x_h and x_v in (9) with continuous variables as follows:

$$\boldsymbol{x_h}, \boldsymbol{x_v}, \boldsymbol{y_h}, \boldsymbol{y_v} \ge \boldsymbol{0}. \tag{10}$$

Proposition 1. The relaxed LP denoted by Eqs. (5)-(8),(10) computes an integral optimal solution and hence solves Problem 1 in polynomial time.

IV. PROOF OF PROPOSITION 1

In general, solving MILPs is NP-hard while LPs can be solved in polynomial time [25]. An LP relaxation of a MILP in general will not yield optimal integral solutions. However, our LP relaxation belongs to a special class of problems that yield integral optimal solutions. This is mainly because of our use of NAI matrices, which are totally unimodular (TU) [26]. A matrix is TU if the determinants of all its square submatrices are in $\{-1, 0, 1\}$ [24]. In this section, we prove that the matrix of all constraints in Eqs. (5)-(8), (10) is TU. It will then follow from the Hoffman-Kruskal Principle [27] that the relaxed LP can directly solve the MILP in polynomial time, since our matrix is TU.

Theorem 1 (Hoffman-Kruskal Principle [27]). *If an integral matrix A is TU, then for an integral vector* **b***, the polyhedron* $\{x \in \mathbb{R}^n \mid Ax \leq b\}$ *has integral coordinates.*

Since the polyhedron has integral coordinates, the resulting optimal solution for the LP defined in the polyhedron is also integral.

Lemma 2. The constraints of the LP formulation in Eqs. (5)-(8),(10) are TU.

Proof. First, we write our problem in Standard Equality Form (SEF) using slack variables to eliminate the inequality constraints. We then rewrite the summations in Eq. 5 using a vector multiplication with 1 and obtain

$$\min \quad \mathbf{1}^T \boldsymbol{y_h} + \mathbf{1}^T \boldsymbol{y_v} \tag{11}$$

s.t.
$$A_H \boldsymbol{x_h} - \boldsymbol{y_h} + \boldsymbol{z_h} = \boldsymbol{0}$$
 (12)

$$A_V \boldsymbol{x_v} - \boldsymbol{y_v} + \boldsymbol{z_v} = \boldsymbol{0} \tag{13}$$

$$\boldsymbol{x_h} + \boldsymbol{x_v} = \boldsymbol{1} \tag{14}$$

$$\boldsymbol{x_h}, \boldsymbol{x_v}, \boldsymbol{y_h}, \boldsymbol{y_v}, \boldsymbol{z_h}, \boldsymbol{z_v} \ge \boldsymbol{0}. \tag{15}$$

The constraints are now of the form Ax = b, where

$$A = \begin{bmatrix} A_H & \overline{\mathbf{0}} & -I & \overline{\mathbf{0}} & I & \overline{\mathbf{0}} \\ \overline{\mathbf{0}} & A_V & \overline{\mathbf{0}} & -I & \overline{\mathbf{0}} & I \\ I & I & \overline{\mathbf{0}} & \overline{\mathbf{0}} & \overline{\mathbf{0}} & \overline{\mathbf{0}} \end{bmatrix},$$
(16)

$$\boldsymbol{b} = \begin{bmatrix} \boldsymbol{0}^T & \boldsymbol{0}^T & \boldsymbol{1}^T \end{bmatrix}^T, \tag{17}$$

$$\boldsymbol{x} = \begin{bmatrix} \boldsymbol{x_h}^T & \boldsymbol{x_v}^T & \boldsymbol{y_h}^T & \boldsymbol{y_v}^T & \boldsymbol{z_h}^T & \boldsymbol{z_v}^T \end{bmatrix}^T, \quad (18)$$

 $\overline{\mathbf{0}}$ is the matrix of zeros and *I* is the identity matrix. Clearly **b** is an integral vector.

To prove A is TU, let us start with the matrix

$$\tilde{A} = \begin{bmatrix} A_H^T & \overline{\mathbf{0}} & I \\ \overline{\mathbf{0}} & A_V^T & I \end{bmatrix}.$$

Each row of the NAI matrices A_H and A_V signifies a directed edge in the graph, with a -1 for the source grid cell (outgoing), a +1 for the sink grid cell (incoming), and 0s otherwise [24].

Due to this construction, \tilde{A} satisfies the following sufficient conditions for TU matrices as outlined in [27].

- 1) All elements are in $\{-1, 0, +1\}$ (all entries in A_V and A_H are either -1, 0, or +1).
- 2) Each column of \hat{A} has at most two non-zero elements (each column of A_H^T and A_V^T will have two non-zero entries, one for the source and one for the sink).
- 3) There exists a partition of the rows of \hat{A} into two disjoint sets T_1 and T_2 such that:
 - (i) If any column of A contains two nonzero entries of the same sign, then one is in a row of T_1 and the other is in a row of T_2 .
- (ii) If any column of \tilde{A} contains two nonzero entries of the opposite sign, then they are both in a row of T_1 or in a row of T_2 .

For \tilde{A} , T_1 consists of all rows in

$$\begin{bmatrix} A_H^T & \overline{\mathbf{0}} & I \end{bmatrix}$$

and T_2 consists of all rows in

$$\begin{bmatrix} \overline{\mathbf{0}} & A_V^T & I \end{bmatrix}$$
.

It follows from [24] that the transpose of \tilde{A} is also TU. We can construct A from \tilde{A}^T by appending columns of the positive and negative signed unit matrices \bar{I} and $-\bar{I}$ where

$$\bar{I} = \begin{bmatrix} I & \bar{\mathbf{0}} & \bar{\mathbf{0}} \\ \bar{\mathbf{0}} & I & \bar{\mathbf{0}} \\ \bar{\mathbf{0}} & \bar{\mathbf{0}} & I \end{bmatrix}$$

Appending these columns preserves the total unimodularity of the resulting matrix, see [24]. Because of this, the matrix A is TU, which completes the proof of Lemma 2.

Since the constraints are TU, it follows from Theorem 1 that the relaxed LP computes optimal integral solutions for the formulated MILP, thereby completing the proof of Proposition 1. We can therefore use efficient polynomial-time LP solvers [25] to solve Problem 1 optimally.

V. TOUR GENERATION

We now address Problem 2 and plan our full coverage path by computing a tour of the ranks obtained from the LP in Section III. We do this by formulating a GTSP to determine a visitation order for the ranks. Using a GTSP formulation offers us the flexibility to use existing approaches to compute the tour [28], [29]. We employ a similar formulation that was proposed in [13] and create an auxiliary graph representing all possible connections between the ranks. The vertices of this graph are grouped into sets, where each set consists of two vertices representing the two directions to cover the rank (e.g., for a horizontal line we can cover it from left to right or right to left). The GTSP aims to minimize the cost of visiting one vertex in every set (traversing each rank in a specific direction).

The edge costs between the vertices in different sets of the auxiliary graph are given by the time to travel between rank endpoints along an obstacle-free transition path. These transition paths are planned using the dynamics of the robot, which depends on the robot's design. For this work, we assume semi-holonomic dynamics where (i) the robot stops to turn in-place with a constant angular velocity, and (ii) the robot has piecewise constant acceleration (when accelerating or decelerating) while travelling in a straight line with a maximum linear velocity. The same dynamics model is used in [11] and [14], which we compare our method against in Section VI. We compute each transition path by constructing a visibility graph within the IOP boundaries and planning the shortest path using an A* search [30]. The resulting path is a sequence of straight lines and intermediary turns, for which the travel time is computed.

VI. RESULTS

In this section, we present our experiments to test the performance of OARP and compare it to two different coverage planning approaches from literature. Finally, we present ROS simulations on an example environment with the Avidbots Neo robot model.



Fig. 5: Comparison of our rank partitioning method with that proposed in [11]: (a) Cell decomposition runtime for each map. (b) The number of ranks determined for each test environment.

Firstly, in Sections VI-A to VI-C, we compare OARP to the heuristic approach from [11]. We chose the heuristic approach for our main comparisons because it produced the least turns and the lowest tour costs in our analysis of the available approaches. While the work in [11] has applied the heuristic approach for both single and multi-robot cases, the rank partitioning step is itself independent of the number of robots, which is similar to OARP. We focus on the single-robot coverage tours in this comparison to analyze how the different rank partitioning methods affect the tour. For this comparison, we use a dataset containing 2D maps of 44 real-world environments obtained from Avidbots and used for their Neo cleaning robot. These environments have a minimum coverage path length ranging from 100 m to around 3700 m, where the minimum coverage path length of an environment is given by (total area)/(tool width l). The maps in this dataset are arranged/labeled by increasing complexity, i.e., the minimum number of ranks required to cover the environment. Due to the confidentiality of these scans, we cannot share this dataset but we have additionally generated a set of anonymized environments for visualization, which are shown in Figs. 1 and 6. Each method is run for 20 trials on each map in the dataset and we compare the results for every stage of the coverage planning process.

A. Decomposition Comparisons

For each map in our dataset, we compared the time it takes for both methods to decompose the environment into cells. The heuristic method decomposes the environment into a preliminary set of coarse rectangular cells [11], while the OARP method simply needs to express the IOP of the environment using a grid overlay. To construct the IOP, we include all grid cells where over 50% of its area is contained within the original environment. The results of the comparison are shown in Fig. 5a, where we observe that as the complexity of the map increases, the cell decomposition time for the heuristic method increases rapidly. In contrast, OARP only requires a grid approximation which can be computed much faster; about 50% faster on average and 5 times faster in some cases. This makes a large impact on the overall planner time, especially in cases where the



Fig. 6: Coverage path of an indoor environment generated using the proposed LP method.

coverage plan would need to be replanned due to perceived changes or uncertainty in the environment. Robustification of our approach to dynamic environment uncertainties is a future research direction.

B. Rank Partitioning Comparisons

We implement our rank partitioning step by expressing the LP using Eqs. (5)-(8), Eq. (10), and the IOP built in the decomposition step. The algorithm was prototyped in Python and we used the free LP solver, GLPK [31] to solve the LP. The coverage tool width was set to 0.8 m to model Avidbots' robot. Fig. 1 shows the rank partitioning of an example map with the horizontal (orange) and vertical (purple) ranks as determined by the LP solver. Solving the LP takes an average of 0.4 seconds across all trials for all maps.

For the heuristic method, we solve the same IOP coverage problem by only including the interior ranks from their method (we ignore the perimeter ranks). The heuristic method is an iterative algorithm, where running for more iterations improves the performance. So, one would need to allow enough iterations to find high quality solutions, but avoid extra iterations after the optimal has been found. In contrast, OARP avoids unnecessary computation by deterministically finding an optimal solution before terminating.

To standardize the comparison, we limit the runtime of the heuristic method's partitioning step to that of OARP's partitioning step (the LP's runtime) for each map. Fig. 5b shows the results of the experiments. As expected, we observe that the number of ranks obtained by OARP is always as good or better than the heuristic method, with the performance gap widening for more complex maps. In general, the heuristic method does quite well, likely due to the tractability of the problem. However, since this problem is tractable and thus solvable in polynomial time, one should choose an optimal solver.

C. Coverage Tour Comparisons

To generate the coverage tour from the ranks, we use the GTSP formulation in Section V to compute a tour that minimizes the transition time between the ranks. Table I documents our choice of parameters used to compute the

TABLE I: Robot parameters used for experiments



Fig. 7: Average % improvement exhibited by the OARP method in comparison with [11] for each map.

transition costs by simulating the robot's coverage tool and dynamics. We used GLNS [28] to solve the formulated GTSP problem for both OARP and the heuristic methods. Fig. 6 shows an example coverage tour for an anonymized environment from our dataset, with the transitions (green lines) between the ranks as determined by the GTSP solver.

We also compared our tours to those obtained using the heuristic method from [11]. Since the ranks are nonoverlapping for both methods, the cost of the tour is related to the number of turns. Fig. 7 shows the comparison results, where we plot the average percent improvement of OARP over the heuristic method for each map on two metrics: a) number of turns, and b) total coverage tour cost. We observe that OARP improves the coverage tours on both metrics for almost all maps. For a small number of maps, we see little to no improvement due to the heuristic method providing the same number of ranks as OARP for those maps. On average for all maps, OARP improved the number of turns by 5.9% and the tour cost by around 2.7%. For some maps, we get a 16.8% improvement in the number of turns, and a 7.6% improvement in total coverage time. From this comparison, we observe the intended trend that less ranks leads to less turns, and less turns leads to shorter coverage paths.

D. Comparison against BCD

We additionally compared the proposed OARP method against the Boustrophedon Cell Decomposition (BCD) method from [14]. For this comparison, we used the dataset from [14] containing over 300 aerial scans of environments with varying number of holes (obstacles). The complexity measure of the maps in this dataset is given by *hole vertices*, i.e., the total number of vertices required to represent the holes in the map (e.g., one triangular hole has 3 vertices). The maximum number of hole vertices in this dataset is 86, whereas in Avidbots' dataset, this number goes up to 900. We do not test BCD on Avidbots' dataset as it would require significant modifications to the BCD implementation, but we



Fig. 8: Comparison results of the coverage tour generated using OARP and the BCD method [14]. The x-axis specifies the number of vertices representing the holes in the maps.



Fig. 9: Simulation results of covering an example environment using the Avidbots Neo robot. The covarage planners used were (a) OARP and (b) the heuristic method from [11].

expect to see a similar but exaggerated trend due to the large number of hole vertices. To perform this comparison, we use the same parameters from [14]; a tool width of 3 m, linear acceleration of $\pm 1 m/s^2$, and maximum velocity of 3 m/s. We also penalize turns by adding turning time to the cost function, where the robot turns in-place with an angular velocity of $30^{\circ}/s$. Fig. 8 shows the results of comparing the cost of the coverage tours generated by OARP and BCD using identical parameters. We observed that OARP outperforms BCD for all maps, and improves the coverage tour by 25.5% on average. We believe that this performance gap is due to OARP's better coverage line placement. We also tested the heuristic method from [11] on this dataset and observed that, while OARP still outperforms [11], the average performance improvement is small (<1%) due to the relative simplicity of the maps in this dataset.

E. ROS Simulation Case Study

In this subsection, we provide a case study of simulating the coverage of an example environment using the Avidbots Neo robot. The simulation was conducted using ROS on an anonymized hybrid environment as shown in Fig. 9. The Neo robot has tricycle dynamics and so coverage was planned with sufficient space at the boundaries to allow comfortable turns. The robot was also equipped with a local planner to replan difficult parts of the coverage plan (e.g. areas with small coverage lines) if necessary. Fig. 9 shows the resulting environment coverage using plans generated by OARP and the heuristic method [11]. We observed that OARP's plan was nearly 13% faster than the heuristic method's plan and 3.3% shorter in path length. We also observed that OARP's plan covered more area of this map than the heuristic method's plan (1.1% more). The reasoning for this is twofold: OARP's path consists of less turns (poor coverage at turns), and OARP's plan required less replans by the local planner. We also attempted to test the plan generated using the BCD planner from [14], but the plan contained many sharp turns that are infeasible for the Neo robot. Recordings of the simulations are also included in our video attachment.

VII. CONCLUSION

In this paper, we proposed the OARP coverage planning method that aims to minimize the number of turns needed to cover the robot's environment. We do this by partitioning the environment into thin axis-parallel ranks using a linear program (LP). We proved that this formulation computes the optimal rank partitioning in polynomial time. We then conducted experiments on maps of real-world environments and showed that OARP always achieves the best rank partition in comparison with the state-of-the-art method. Furthermore, OARP is also shown to have around 6% lesser turns and 3% shorter coverage tours on average than the other method.

While OARP can be used for any environment, applying the axis-parallel constraint may result in "stair-case" paths to cover narrow areas with non-axis-parallel or curved boundaries. A future research direction is relaxing this constraint to improve coverage of such areas. We also aim to leverage the performance of OARP to develop an online turn-minimizing coverage planner for uncertain dynamic environments.

ACKNOWLEDGMENT

This work was supported by Canadian Mitacs Accelerate Project IT16435 and Avidbots Corp, Kitchener, ON, Canada.

REFERENCES

- 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.
- [2] C. Hofner and G. Schmidt, "Path planning and guidance techniques for an autonomous mobile cleaning robot," *Robotics and Autonomous Systems*, vol. 14, no. 2, pp. 199–212, 1995.
- [3] I. A. Hameed, "Intelligent coverage path planning for agricultural robots and autonomous machines on three-dimensional terrain," *Journal of Intelligent and Robotic Systems: Theory and Applications*, vol. 74, no. 3-4, pp. 965–983, 2014.
- [4] S. Song, D. Kim, and S. Jo, "Online coverage and inspection planning for 3D modeling," *Autonomous Robots*, vol. 44, no. 8, pp. 1431–1450, 2020.
- [5] W. Jing, D. Deng, Z. Xiao, Y. Liu, and K. Shimada, "Coverage Path Planning using Path Primitive Sampling and Primitive Coverage Graph for Visual Inspection," in *IEEE/RSJ International Conference* on Intelligent Robots and Systems (IROS), 2019, pp. 1472–1479.
- [6] I. Z. Biundini, M. F. Pinto, A. G. Melo, A. L. M. Marcato, L. M. Honório, and M. J. R. Aguiar, "A Framework for Coverage Path Planning Optimization Based on Point Cloud for Structural Inspection," *Sensors*, vol. 21, no. 2, p. 570, 2021.
- [7] B. Nasirian, M. Mehrandezh, and F. Janabi-Sharifi, "Efficient Coverage Path Planning for Mobile Disinfecting Robots Using Graph-Based Representation of Environment," *Frontiers in Robotics and AI*, vol. 8, p. 4, 2021.
- [8] A. Das, M. Diu, N. Mathew, C. Scharfenberger, J. Servos, A. Wong, J. S. Zelek, D. A. Clausi, and S. L. Waslander, "Mapping, Planning, and Sample Detection Strategies for Autonomous Exploration," *Journal of Field Robotics*, vol. 31, no. 1, pp. 75–106, 2014.
- [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] H. Choset and P. Pignon, "Coverage Path Planning: The Boustrophedon Cellular Decomposition," *Field and Service Robotics*, pp. 203– 209, 1998.
- [11] I. Vandermeulen, R. Groß, and A. Kolling, "Turn-minimizing multirobot coverage," in 2019 IEEE International Conference on Robotics and Automation (ICRA), 2019, pp. 1014–1020.
- [12] R. Bormann, F. Jordan, J. Hampp, and M. Hägele, "Indoor Coverage Path Planning: Survey, Implementation, Analysis," in 2018 IEEE International Conference on Robotics and Automation (ICRA), May 2018, pp. 1718–1725.
- [13] 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.
- [14] 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*, ser. Springer Proceedings in Advanced Robotics. Singapore: Springer, 2021, pp. 277–290.
- [15] M. Torres, D. A. Pelta, J. L. Verdegay, and J. C. Torres, "Coverage path planning with unmanned aerial vehicles for 3D terrain reconstruction," *Expert Systems with Applications*, vol. 55, pp. 441–451, 2016.
- [16] J. Jin and L. Tang, "Optimal Coverage Path Planning for Arable Farming on 2D Surfaces," *Transactions of the ASABE*, vol. 53, no. 1, pp. 283–295, 2010.
- [17] T. M. Cabreira, P. R. Ferreira, C. D. Franco, and G. C. Buttazzo, "Grid-Based Coverage Path Planning With Minimum Energy Over Irregular-Shaped Areas With UAVs," in 2019 International Conference on Unmanned Aircraft Systems (ICUAS), 2019, pp. 758–767.
- [18] 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.
- [19] 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.
- [20] M. Theile, H. Bayerlein, R. Nai, D. Gesbert, and M. Caccamo, "UAV Coverage Path Planning under Varying Power Constraints using Deep Reinforcement Learning," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2020, pp. 1444–1449.
- [21] P. T. Kyaw, A. Paing, T. T. Thu, R. E. Mohan, A. Vu Le, and P. Veerajagadheswar, "Coverage Path Planning for Decomposition Reconfigurable Grid-Maps Using Deep Reinforcement Learning Based Travelling Salesman Problem," *IEEE Access*, vol. 8, pp. 225945– 225956, 2020.
- [22] K. G. S. Apuroop, A. V. Le, M. R. Elara, and B. J. Sheu, "Reinforcement Learning-Based Complete Area Coverage Path Planning for a Modified hTrihex Robot," *Sensors*, vol. 21, no. 4, p. 1067, 2021.
- [23] A. Krishna Lakshmanan, R. Elara Mohan, B. Ramalingam, A. Vu Le, P. Veerajagadeshwar, K. Tiwari, and M. Ilyas, "Complete coverage path planning using reinforcement learning for Tetromino based cleaning and maintenance robot," *Automation in Construction*, vol. 112, p. 103078, 2020.
- [24] G. L. Nemhauser, *Integer and Combinatorial Optimization*, ser. Wiley-Interscience Series in Discrete Mathematics and Optimization. New York: Wiley, 1999.
- [25] N. Karmarkar, "A new polynomial-time algorithm for linear programming," *Combinatorica*, vol. 4, no. 4, pp. 373–395, 1984.
- [26] L. Pitsoulis, K. Papalamprou, G. Appa, and B. Kotnyek, "On the representability of totally unimodular matrices on bidirected graphs," *Discrete Mathematics*, vol. 309, no. 16, pp. 5024–5042, 2009.
- [27] A. J. Hoffman and J. B. Kruskal, "Integral Boundary Points of Convex Polyhedra," in 50 Years of Integer Programming 1958-2008: From the Early Years to the State-of-the-Art. Berlin, Heidelberg: Springer, 2010, pp. 49–76.
- [28] 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.
- [29] C. E. Noon and J. C. Bean, "An Efficient Transformation Of The Generalized Traveling Salesman Problem," *INFOR: Information Systems* and Operational Research, vol. 31, no. 1, pp. 39–44, 1993.
- [30] K. J. Obermeyer and Contributors, "VisiLibity: A C++ Library for Visibility Computations in Planar Polygonal Environments," 2008.
- [31] A. O. Makhorin, "GLPK GNU Project Free Software Foundation (FSF)," https://www.gnu.org/software/glpk/.