2022 IEEE 18th International Conference on Automation Science and Engineering (CASE)
August 20-24, 2022. Mexico City, Mexico

2022 IEEE 18th International Conference on Automation Science and Engineering (CASE) | 978-1-6654-9042-9/22/$31.00 ©2022 IEEE | DOI: 10.1109/CASE49997.2022.9926483

Complete, Decomposition-Free Coverage Path Planning

Tushar Kusnur and Maxim Likhachev

***Abstract*— Coverage Path Planning (CPP) requires planning collision-free paths for a robot that observes all reachable points of interest in an environment. Most popular CPP approaches are hierarchical and decomposition-based, involving three steps: (1) decomposing the environment into sub-regions (rectangles or polygons) that simplify the generation of space-filling paths, (2) determining a visitation order over these sub-regions via graph search or a Traveling Salesman Problem (TSP) solver, and (3) generation of space-filling paths in each sub-region. This approach requires significant processing of the environment and the availability of suitable TSP solvers. Furthermore, step (1) can sometimes fail in non-convex environments or lead to "over-decomposition" in cluttered environments. To the best of our knowledge, existing *decomposition-free* approaches are heuristic or random, and therefore typically inefficient and probabilistically complete. We present a resolution-complete decomposition-free coverage path planner that effectively folds steps (1) and (2) above into a single online search routine, making it significantly easier to integrate into existing robot architectures and applicable to a larger set of environments. Our approach leverages a precomputed library of space-filling coverage patterns and automatically determines where to apply them. We evaluate our approach on a variety of environments to demonstrate its benefits and provide an open-source implementation at** https://github.com/ktushar14/cdf_cpp.

I. Introduction

Coverage Path Planning (CPP) is the problem of computing a feasible, collision-free path for a robot that passes within some sensor footprint of all reachable points of interest in an environment [1], [2]. Several real-world tasks can be cast as 2D coverage problems, including aerial surveys [3], [4], post-disaster assessment [5], and agricultural operations [6], [7]. The CPP problem is computationally expensive—it is related to the covering salesman problem, where an agent must visit a neighborhood of each city [8], which is in turn related to the NP-Complete Traveling Salesman Problem (TSP) [9]. Further, real robots like unmanned aerial vehicles (UAVs) often have various constraints on movement, such as limits on turning radius, translational and rotational speed, and sensor footprint field-of-view [10], [11].

Prior work (Sec. II) shows that most popular CPP solutions are hierarchical, and typically involve three steps: (1) Decomposition: This divides the environment into mutually exclusive and exhaustive sub-regions, wherein it is simple to generate space-filling or sweeping coverage paths. These paths do not necessarily account for obstacles and are composed of simple motions—such as straight lines or circles—to provide simplicity in control. (2) Sub-region traversal: This involves determining a visitation order over these sub-regions, typically via a TSP solver. (3) Generation of space-filling paths: This involves generating sweeping coverage patterns in each sub-region, such as back and forth Boustrophedon* motions, spiral patterns, and others.

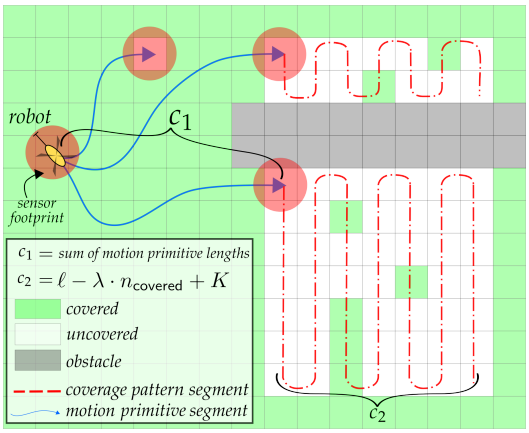


Fig. 1: A figure illustrating three out of several possible paths our planner might choose in a toy environment, reasoning over combinations of frontier-seeking motions and space-filling patterns. Full paths include both the motion primitive segment *and* the coverage pattern segment if present.

Decomposition strategies are often tailored to specific objectives and come with guarantees of satisfying them, but at the cost of significant effort spent in processing the environment. However due to the geometric nature of decomposition strategies, they can fail in among certain types of environment geometries. They can also lead to over-decomposition in non-convex environments [2], [12], [13], amended by post-processing step to merge adjacent sub-regions. Furthermore, decomposition almost always follows solving a TSP or similar [14], [12], [15], [16].

Few approaches require no environment decomposition (or even representation) which Choset categorized as "heuristic and randomized approaches" [2]. These arguably require the least development effort, but can be inefficient—they do not search for paths but randomly select local behaviors or templates, relying on probabilistic completeness [17], [18]. However they do not suggest methods to combine these templates to achieve full coverage of a target region [2].

Our key insight is that we can fold the decomposition and subregion-traversal steps into a single search routine that can be called repeatedly till coverage is complete (Sec. III). To that end, our contribution is an online, **decomposition-free, resolution-complete coverage path planner**. Our

The authors are with the Robotics Institute in the School of Computer Science at Carnegie Mellon University, Pittsburgh, USA: {tkusnur, maxim}@cs.cmu.edu. This work was supported by the ONR grant N00014-18-1-2775.

*Boustrophedon means back-and-forth, like the motion of an ox ploughing a field.

planner uses a grid-based representation of the environment and provides continuous coverage paths. We fill a gap in previous work on CPP by providing completeness guarantees with little environment processing effort (see Fig. 2) and applicability to a variety of environments. There is of course no free lunch—to achieve this, our planner leverages a precomputed set of space-filling patterns. The planner reasons over these patterns and determines suitable parts of the environment where they can be applied. We draw inspiration from node selection strategies in frontier-based exploration [19] to enable this behavior.

In Sec. IV, we demonstrate our method on a variety of environments and compare quantitatively with a simpler and complete decomposition-free baseline that employs frontier-based exploration. We also qualitatively compare against [12] to show that our approach yields similar, intuitive paths without the decomposition and sub-region traversal steps. However our method does not similarly guarantee optimality with respect to coverage objectives such as turn- or distance-minimization. We summarize our work and discuss future research problems in Sec. V.

II. Related Work

We categorize prior work into decomposition-based and decomposition-free methods and briefly touch upon frontier-based exploration planning. We point the reader to [1], [2] for extensive surveys on coverage path planning. Multi-robot coverage is beyond the scope of this paper.

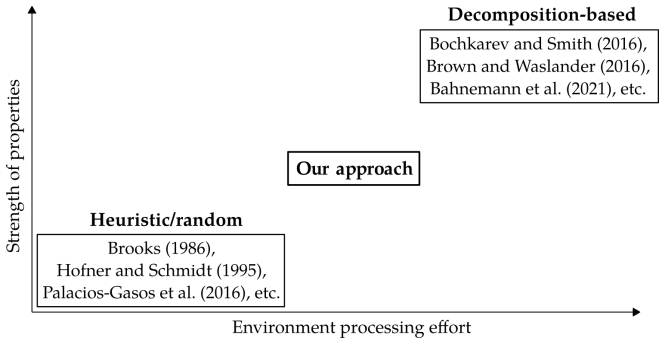


Fig. 2: Our approach contextualized within representative prior work roughly clustered by amount of explicit environmental processing effort involved and strength of properties provided.

Decomposition-based approaches. These can be categorized into exact (continuous) and inexact (grid-based) cellular decomposition (CD). Exact CD involves partitioning the free space of the environment into simple, non-overlapping regions called cells. Early exact CD approaches include Trapezoidal, Boustropheon, and Morse-based decomposition. More recent approaches exist that optimize for specific objectives. These include minimizing turns in coverage paths [20], minimizing coverage path length [14], and creating intuitive, human-like sub-regions [12]. While typically separate from CPP literature, there also exist closely related "room segmentation" strategies that operate specifically on indoor floor plans, of which Bormann et al. provide a thorough survey [21]. Among exact decomposition strategies, the simple trapezoidal decomposition can fail in non-polygonal environments [22], [23], and the more general Morse-based decomposition fails in rectilinear environments [24].

Inexact or grid-based approaches discretize the environment into uniform grid cells, typically as large as the robot's coverage footprint, and employ various space-filling methods with backtracking strategies such as spanning trees and spiral patterns [25], [26]. However these approaches do not take advantage of smooth sweeping motions such as the Boustrophedon pattern, except for a recent paper utilizes it with turn-minimizing cellular decomposition [15].

Decomposition-free approaches. Early approaches like these are heuristic algorithms where the robot is equipped with a set of behaviors such as wall-following, obstacle-avoidance, etc. [17], [27], [28], [29]. There are also commercial floor-cleaning robots based on this approach (the RC3000 by Karcher, Trilobite by Electrolux, and early versions of the Roomba by iRobot) [1], [30]. These approaches do not guarantee completeness but have advantages from a cost/benefit standpoint—they are easy to implement and do not require costly localization sensors [31]. Our work can be thought of as a method to stitch together such local behaviors to enable complete coverage.

Frontier-based exploration planning In frontier-based exploration, a robot repeatedly moves toward the closest point on the frontier—which is the set of cells at the border between explored and unexplored parts of the environment (covered and uncovered in our case) [19], [32], [33], [34], [35], [4], [36]. We employ a frontier-based search to determine where space-filling coverage patterns should be applied (details in Sec. III).

III. Method

We assume we are given an occupancy grid map of an uncovered 2D environment, the start configuration of the robot, and parameters relating to constraints on the robot's motion and sensor footprint. We also assume access to a library of precomputed space-filling coverage patterns. We require the robot to observe all reachable points of interest in the environment. Typical desirable properties of coverage paths include simple motions and minimal overlaps and number of turns. The precomputed coverage patterns can individually be optimal with respect to such objectives, but our approach does not provide any guarantees on optimality of the overall path.

A. Framework

Control flow. Our framework relies on repeated queries to a search-based planner which returns a feasible, collision-free coverage path for the robot. In Alg. 1, we present pseudocode for our overall framework We begin coverage by calling PathPlanner(s_{start}, $\mathcal{M}$) to obtain and execute a path π and then repeatedly do this until the planner does not return a path (Lines 2–8) At this point (line 4), coverage is complete because the planner is guaranteed to return a path as long as reachable uncovered cells exist in the environment.

Algorithm 1 Framework overview—MAIN procedure

Inputs: s_{start} (robot start configuration), $\mathcal{M}$ (grid map)

1: **procedure** MAIN($s_{\text{start}}, \mathcal{M}$)
2: **repeat**
3: $\pi \leftarrow$ PATHPLANNER($s_{\text{start}}, \mathcal{M}$) ▷ See Alg. 2.
4: **if** π is NULL **then**
5: done ▷ Coverage complete; terminate
6: Execute π and update $\mathcal{M}$
7: $s_{\text{start}} \leftarrow$ last state in π ▷ Assuming perfect execution
8: **until** done

Search-based planning on lattice graphs. As is common in search-based planners operating over grids, we discretize the environment into grid cells with a uniform resolution. The planner is resolution-complete with respect to this underlying grid. Each cell in the grid falls into one of three categories: free & uncovered, free & covered, and blocked. The planner searches for a path over a finite, discrete lattice graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ where vertices/nodes $\mathcal{V}$ represent robot configurations connected by edges $\mathcal{E}$ representing continuous motions feasible by the robot [37], [38]. These paths are always start and end on a lattice state, and can either be motion primitives or space-filling coverage paths (details follow).

We require a set of motion primitives $\mathcal{A}$, which are simply short, continuous, feasible motions, that connect pairs of lattice states. This is standard practice [37], [38] and allows our approach to operate on a variety of robots given their kinematic and dynamic constraints.

We also require access to a set of precomputed space-filling coverage patterns $\mathcal{P}$ which completely cover areas of various simple shapes, each of which is a continuous path centered at the origin. In this paper, we run experiments only using Boustrophedon patterns that cover rectangular areas of various sizes, but any set of patterns that are feasible for the robot can be used in our framework.

B. Details—Procedure PATHPLANNER

A frontier cell $f \in \mathcal{F}$ is an uncovered cell in $\mathcal{M}$ that has at least one neighboring cell that is covered. Standard frontier-based exploration [19] is an iterative approach, where in each iteration the robot moves to the closest frontier cell, covers it, and repeats this until coverage is complete. Our approach is similar, except that in each iteration, the planner reasons about all combinations of frontier-seeking motions and the several coverage patterns available to it at each frontier cell.

Multi-Goal search. At any arbitrary point in time during coverage, let s_{start} be the node corresponding to the robot's current location. We also define a *frontier node* as a node that corresponds to a robot configuration whose position (x_R, y_R) is a frontier cell in $\mathcal{M}$. The graph representing the environment contains a set of frontier nodes $f_i \in \mathcal{F},\ i = 1, \ldots, |\mathcal{F}|$[†]. To make progress in coverage, the robot must visit (explore) a frontier node and possibly execute a coverage pattern. Since there are multiple frontier nodes, this can be viewed as a multi-goal motion planning problem where each frontier node is a potential goal.

[†]For brevity, we represent the sets of both frontier cells and frontier nodes as $\mathcal{F}$.

Algorithm 2 Method details—Multi-goal Dijkstra's search

1: **procedure** PATHPLANNER($s_{\text{start}}, \mathcal{M}$)
2: $g(s_{\text{start}}) \leftarrow 0$
3: OPEN $\leftarrow$ OPEN $\cup \{s_{\text{start}}, g(s_{\text{start}})\}$
4: **while** OPEN not empty **do**
5: $x \leftarrow$ OPEN.MIN()
6: SUCCS $\leftarrow \emptyset$ // List of successor nodes
7: COSTS $\leftarrow \emptyset$ // List of corresponding edge costs
8: **if** $x == G_{\text{imaginary}}$ **then**
9: $\pi \leftarrow$ Backtrack from $G_{\text{imaginary}}$ to s_{start}
10: **return** π // Path found
11: **else**
12: EXPAND $(x, \mathcal{M}, \text{SUCCS}, \text{COSTS})$
13: **for** successor $s' \in$ SUCCS and cost $c \in$ COSTS **do**
14: **if** $g(s') > g(s) + c$ **then**
15: $g(s') \leftarrow g(s) + c$
16: $bp(s') \leftarrow s$
17: OPEN $\leftarrow$ OPEN $\cup \{s', g(s)\}$
18: **return** NULL // No path found

19: **procedure** EXPAND($x, \mathcal{M}$, SUCCS, COSTS)
20: **if** x is a frontier node **then**
21: // Apply coverage patterns at frontier cell
22: **for** collision-free pattern p in $\mathcal{P}$ **do**
23: APPLYPATTERN $(x, p, \mathcal{M}, \text{SUCCS}, \text{COSTS})$
24: // Only cover frontier cell (no pattern applied)
25: SUCCS $\leftarrow$ SUCCS $\cup\ G_{\text{imaginary}}$
26: COSTS $\leftarrow$ COSTS $\cup\ K - \lambda \cdot 1$
27: **else**
28: // Apply motion primitives at standard cell
29: **for** collision-free motion primitive a in $\mathcal{A}$ **do**
30: APPLYMPRIM $(x, a, \mathcal{M}, \text{SUCCS}, \text{COSTS})$

31: **procedure** APPLYPATTERN($x, p, \mathcal{M}$, SUCCS, COSTS)
32: SUCCS $\leftarrow$ SUCCS $\cup\ G_{\text{imaginary}}$
33: COSTS $\leftarrow$ COSTS $\cup \{\ell^p - \lambda * n_c^p + K\}$

34: **procedure** APPLYMPRIM($x, a, \mathcal{M}$, SUCCS, COSTS)
35: SUCCS $\leftarrow$ SUCCS $\cup$ {node representing state after a applied at x}
36: COSTS $\leftarrow$ COSTS $\cup \{a.\ell\}$

We introduce an *imaginary goal node* $G_{\text{imaginary}} \in \mathcal{V}$ in $\mathcal{G}$ —this node does not correspond to any real location, but is only present to better formulate our search. Multi-goal Dijkstra's search is equivalent to standard Dijkstra's search [39] from s_{start} to $G_{\text{imaginary}}$. As in standard graph search terminology [40], (1) the g-value of a node is its cost-to-come from node s_{start}, and (2) OPEN is a priority queue of nodes (in our case, implemented as a min-heap), where priorities associated with each node are their g-values in the case of Dijkstra's search.

In Alg. 2, we initialize the search by inserting s_{start} into OPEN with a g-value of 0 (Lines 2–3). We then proceed as in standard Dijkstra's search, expanding nodes from OPEN in a best-first fashion. The steps involved in expanding a node are key to our approach. Suppose at a point during the search, node x is obtained (popped) from OPEN. We perform one of the following three steps:

- If x is not a frontier node, we apply motion primitives $\mathcal{A}$ applicable at x to generate successors SUCCS and costs COSTS (Lines 17–20).
- If x is a frontier node, we apply coverage patterns $\mathcal{P}$ (Lines 10–13). The successor at the end of each applied pattern is $G_{\text{imaginary}}$ and the costs incorporate distance traversed

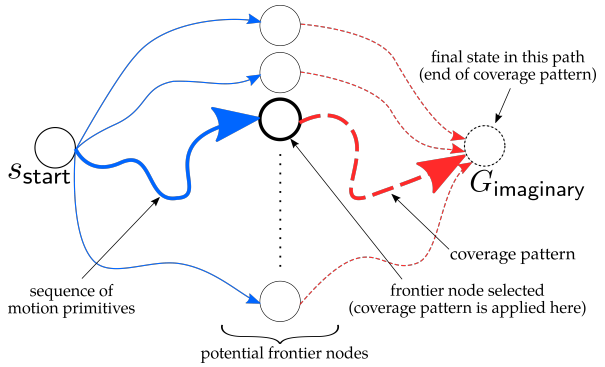


Fig. 3: Multi-goal Dijkstra search tree. For illustration, we show only a single coverage pattern at each frontier node, but there are often multiple in reality.

by the pattern as well as the number of cells covered. (We elaborate on this below.) We also add a successor corresponding to not executing any pattern at the frontier node (Lines 14–16). This provides the option of simply executing a frontier-seeking motion.

- The third and last option for x is that it is node $G_{\text{imaginary}}$, in which case we terminate the search, backtrack from $G_{\text{imaginary}}$ to s_{start}, and return the resulting path (Lines 7–9).

Edges outgoing from a frontier node incorporate the cost of the robot traveling and covering cells using a particular pattern applied at that frontier node in their each of its successor nodes. Edges incoming to a frontier node incorporate the cost of the robot traveling to that frontier node. In this way, for the given set of patterns and motion primitives, the search computes a coverage path for the robot to execute in a query. A visual of this is presented in Fig. 3 for clearer understanding. The rest of the algorithm (Lines 21–25) is identical to Dijkstra's search, where g-values of successor nodes are updated or inserted into `OPEN` whenever better paths to them are found.

C. Details—Procedure ApplyPattern

We now detail the cost function used when applying a coverage pattern. Say executing a pattern p results in traveling a distance ℓ^p and covering n_c^p number of cells in $\mathcal{M}$ that were uncovered before executing the pattern. The cost function we use for executing this pattern is:

$$\mathsf{cost}(x, p, \mathcal{M}) = \ell^p - \lambda \cdot n_c^p + K \tag{1}$$

Here, $\lambda > 1$ is a user-defined parameter that controls how much to weigh covering n_c^p cells at the expense of traveling a distance ℓ^p. The larger λ is, the more likely the planner is to apply coverage patterns, but this might increase non-working distance as it might result in traveling more often over cells that are already covered. The constant K ensures that this cost is always non-negative. Note that this does not distort the solution by incentivizing shorter paths, as this constant is not added to every edge in the graph. As mentioned before, the successor node of applying a pattern is always $G_{\text{imaginary}}$ (Line 37).

D. Details—Procedure ApplyMPrim

Finally, in ApplyMPrim we use the length of the primitive as the cost, to be consistent with ℓ^p above. For UAV-like robots, motion primitives practical for planning are simple, short motions starting and ending at lattice states. From a coverage perspective, frontier-seeking motions will contain such short primitives to approach a frontier cell, but with sweep-coverage patterns we can more easily have long, smooth paths.

IV. Evaluation

A. Experiment details

Sweep-coverage patterns. We precompute and store boustrophedon coverage patterns that each cover a rectangular area of some width and height. Each pattern consists of long, straight-line segments and circular arc segments connecting them. We discretize our map into cells of dimension D and we choose $D = 2R$ where R is the radius of turns in the boustrophedon patterns we use in experiments. This is done so that each pattern exhaustively covers said rectangular areas without any overlapping paths. Different coverage patterns for different settings of R, D can be precomputed and just as easily be used in our approach, provided we have access to ℓ^p and n_c^p (Sec. III-C).

Environments. We evaluate our approach in simulation across multiple environments, namely, rooms and walls with random gaps (E1), corridors and rooms (E2), and city-like maps from the MovingAI benchmark[‡] (E3). Several prior works focus on indoor scenarios due to heavy presence of non-convex and cluttered shapes and use special methods to decompose these environments [12], [15], [21], [33]. While we show the completeness of our decomposition-free approach on such environments, our approach is applicable to any environment that can be represented as a uniformly discretized grid map. Dimensions of the environments we consider are 100 times the sensor footprint diameter D (small), and 250 times D (large). From previous work on aerial coverage [4], we know that a typical sensor footprint diameter for a helicopter-like UAV is $\sim$ 30m. Thus our largest environments may represent areas up to $\sim$ 7500m $\times$ 7500m. Fig. 4 shows representative examples from these environment categories.

For experiments on environments of $100D \times 100D$ units, we use use coverage patterns covering rectangles of sizes up to $30D \times 30D$ units in increments of D units—a dense set of space-filling patterns. We include separate patterns beginning execution at each of the four corners of the corresponding rectangular area, which makes over 3000 patterns. For environments of $250D \times 250D$ units, we uniformly sample up to 25 patterns of sizes up to $160D \times 160D$ units to save on computational effort.

[‡]https://movingai.com/benchmarks/grids.html

Finally, as seen in Fig. 6, we also use two handcrafted environments to compare with decomposition-based baselines[§].

Robot. While our approach is general and works with any type of robot, for experiments we model a planar UAV with a fixed, circular sensor footprint, the configuration of which consists of its position and yaw: $(x, y, \gamma) \in SE(2)$ [4]. We use simple unicycle dynamics [41] to generate motion primitives: $\dot{x} = v\cos\theta,\ \dot{y} = v\sin\theta,\ \dot{\theta} = \omega$. Further, we have upper bounds on the robot's translational speed $v \leq v_{\mathsf{max}} := 8\mathrm{m/s}$ and rotational speed $\omega \leq \omega_{\mathsf{max}} := 0.14/\mathrm{s}$ (these values are borrowed from previous work [4]). Lastly, the robot's sensor has a circular footprint on the ground of radius $\frac{D}{2}$ centered at the robot's position, meaning the robot's footprint covers one cell at a time. We measure performance in terms of the distance traveled while covering the environment, total planning times, and total path execution times.

B. Decomposition-free comparison.

Few, early decomposition-free approaches exist, which are are heuristic or random. Running random coverage planning as in [18] is highly inefficient in the environments we consider, and so we do not include it in our experiments. For example, for rooms in E1 with very narrow gaps, the robot takes a very long time to move to another room when using random actions. We implement a simple, *complete* decomposition-free baseline: a frontier-based approach (akin to vanilla frontier-based exploration), which we denote this by "frontier-based coverage" (FBC). This is equivalent to our approach *without* any preocmputed space-filling patterns (i.e., Alg. 2 without lines 21–23). In this approach, the robot repeatedly covers the frontier cell closest to it until coverage is complete.

We present results for two measures of efficiency: (1) The amount of coverage versus distance traveled by the robot, and (2) total planning times and execution times. Experiments were run on an Intel® Core™ i7-6700 CPU @ 3.40GHz × 8 and approaches implemented in C++ and compiled using g++ 9.4 with the -O3 optimization flag. For maps of size $100D \times 100D$, for both environment types E1 and E2, and we run both our framework and the baseline until complete coverage is achieved for 20 trials. For maps of size $250D \times 250D$ of environment types E1 and E2 and that of size $256D \times 256D$ of environment type E3, we terminate our framework when 90% coverage is achieved by it. At this point of time, FBC has covered a significantly smaller area of the environment, and we terminate it in the interest saving computational effort. We would require FBC to run to completion to compute *total* planning and execution times, and so we do not include them for this set of environments. We run 5 trials for maps of size $250D \times 250D$ of environment types E1 and E2. Note that we use axis-aligned boustrophedon (back-and-forth) patterns as sweep coverage patterns available to the planner. These patterns are highly suitable for environment types E1 and E2, but not particularly for E3, as the obstacles are less structured. We run experiments on maps of type E3 (4 trials) to illustrate that even if coverage patterns are not specifically tailored to the environment, our framework still shows significant benefits over FBC.

[§]The maps and open-source code for these baselines were not available at the time of working on this paper. So we manually replicated two environments for a qualitative comparison.

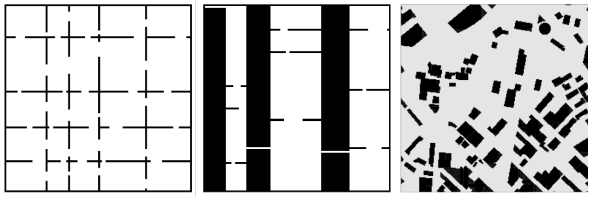


Fig. 4: Representative examples of environments used in experiments. Starting from top-left moving clockwise: E1: walls + gaps, E2: corridors + rooms, and E3: city maps (Boston_1_256 map shown here) from the MovingAI benchmark.

Our framework (plots in red) significantly outperforms FBC (plots in blue) in all cases with respect to the distance traveled to complete coverage, as seen in subfigures A, B, and C in Fig. 5. We also demonstrate much smaller planning and execution times as the planner is queried significantly fewer times in our approach (Tables **I** and **II**). We post-process plans to compute total execution times assuming constant velocities for the motion primitives and coverage patterns ranging between 2m/s and 8m/s (same across both approaches).

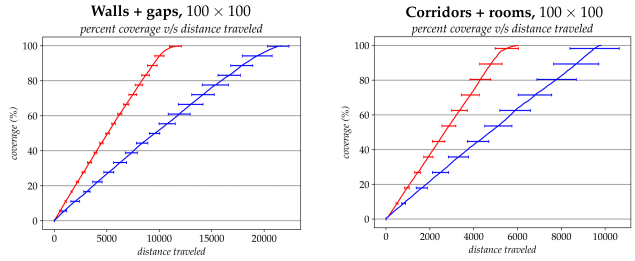
In maps of size $250D \times 250D$ for environment types E1 and E2 we see steep increases in coverage (vertical red increases in the plots) whenever a coverage pattern is executed. The more gentle increases in coverage either correspond to coverage patterns overlapping with previous paths or several consecutive frontier-seeking motions. This occurs because, as mentioned earlier, we uniformly sample coverage patterns to keep planning times reasonable, and as a result, there does not exist a coverage pattern that exhaustively covers every possible rectangular area the planner might encounter.

C. Decomposition-based comparison.

In Fig. 6 we present a qualitative comparison of coverage paths computed by our approach with representative examples from Brown and Waslander's Constriction Decomposition method [12] and the linear-programming based turn-minimizing approach by Ramesh et al. [15]. Similar to us, Brown and Waslander do not claim any overall optimality guarantees. Their approach is based on geometrically computing "constriction points" in the environment, which correspond to areas like openings near doors or corridors in indoor scenarios. We observe that in our approach as well, constriction points naturally emerge as frontier nodes near such areas. Unlike the approach by Ramesh et al., we do not claim any overall optimality guarantees, which is where the environment partitioning effort of that baseline pays off. However our approach is complete without needing any environment partitioning/decomposition and yields intuitive coverage paths.

TABLE I: Planning and execution time results $100D \times 100D$

Metrics	E1		E2	
	Ours	FBC	Ours	FBC
Total planning times (s)	$\mathbf{222 \pm 37}$	2931 ± 449	$\mathbf{57 \pm 12}$	551 ± 132
Total execution times (s)	$\mathbf{12530 \pm 1144}$	29825 ± 658	$\mathbf{4842 \pm 424}$	17946 ± 943



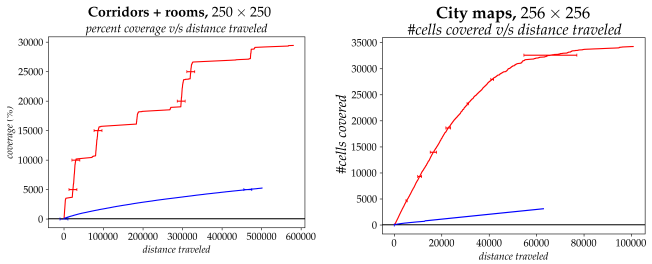


Fig. 5: All results including plots of coverage versus distance traveled for all environments (**A**, **B**, **C**) and total planning and execution times for environments of size $100D \times 100D$ (Tables I and II), as well as plots for larger environments of size $250D \times 250D$.

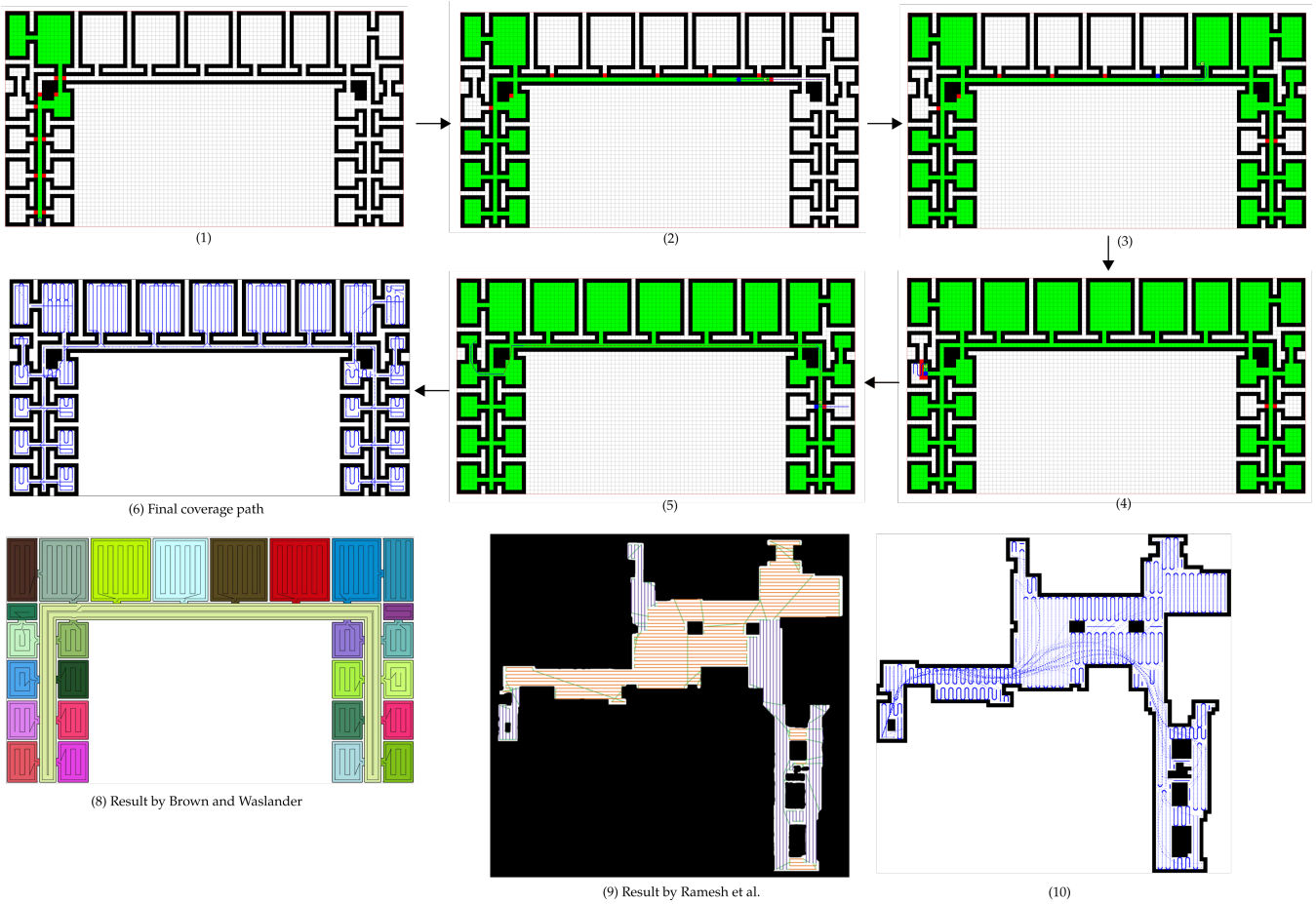


Fig. 6: Comparison with Brown and Waslander's Constriction Decomposition method [12] on and the turn-minimizing approach by Ramesh et al. [15] on representative environments. Subfigures 1–6 show our method on an indoor lab environment and subfigure (7) shows Brown and Waslander's result on a similar environment. Subfigure 9 shows our method on another indoor environment and subfigure (8) shows the turn-minimal solution by Ramesh at al. on a similar environment. (We handcrafted these environments in an occupancy grid format and so they are not fully identical to those used in the baselines.)

V. Conclusions

We present a decomposition-free approach to complete coverage path planning. The search reasons over paths consisting of motion primitives and precomputed space-filling coverage patterns and automatically determines where in the environment a space-filling pattern should be applied. We evaluate our approach in a variety of environments and compare it qualitatively and quantatively with appropriate baselines. In large environments, we sample patterns to prevent slow performance and show that it is still effective and does not require decomposition. For the future, in order to improve scalability, we wish to look into dynamically generating these space-filling patterns online by geometric or data-driven controllers.

REFERENCES

[1] 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] H. Choset, "Coverage for robotics–a survey of recent results," *Annals of mathematics and artificial intelligence*, vol. 31, no. 1, pp. 113–126, 2001.

[3] K. Shah, G. Ballard, A. Schmidt, and M. Schwager, "Multidrone aerial surveys of penguin colonies in antarctica," *Science Robotics*, vol. 5, no. 47, p. eabc3000, 2020.

[4] T. Kusnur, S. Mukherjee, D. M. Saxena, T. Fukami, T. Koyama, O. Salzman, and M. Likhachev, "A planning framework for persistent, multi-uav coverage with global deconfliction," in *2019 International Conference on Field and Service Robotics (FSR)*, 2019.

[5] A. Nedjati, G. Izbirak, B. Vizvari, and J. Arkat, "Complete coverage path planning for a multi-UAV response system in post-earthquake assessment," *Robotics*, vol. 5, no. 4, p. 26, 2016.

[6] T. Oksanen and A. Visala, "Coverage path planning algorithms for agricultural field machines," *Journal of field robotics*, vol. 26, no. 8, pp. 651–668, 2009.

[7] P. Maini, B. M. Gonultas, and V. Isler, "Online coverage planning for an autonomous weed mowing robot with curvature constraints," *IEEE Robotics and Automation Letters*, 2022.

[8] E. M. Arkin and R. Hassin, "Approximation algorithms for the geometric covering salesman problem," *Discrete Applied Mathematics*, vol. 55, no. 3, pp. 197–218, 1994.

[9] C. H. Papadimitriou, "The euclidean travelling salesman problem is np-complete," *Theoretical computer science*, vol. 4, no. 3, pp. 237–244, 1977.

[10] T. Kusnur, D. M. Saxena, and M. Likhachev, "Search-based planning for active sensing in goal-directed coverage tasks," in *2021 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2021, pp. 15–21.

[11] S. Arora and S. Scherer, "Pasp: Policy based approach for sensor planning," in *2015 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2015, pp. 3479–3486.

[12] S. Brown and S. L. Waslander, "The constriction decomposition method for coverage path planning," in *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2016, pp. 3233–3238.

[13] S. Gholami Shahbandi and M. Magnusson, "2d map alignment with region decomposition," *Autonomous Robots*, vol. 43, no. 5, pp. 1117–1136, 2019.

[14] R. Mannadiar and I. Rekleitis, "Optimal coverage of a known arbitrary environment," in *2010 IEEE International conference on robotics and automation*. IEEE, 2010, pp. 5525–5530.

[15] M. Ramesh, F. Imeson, B. Fidan, and S. L. Smith, "Optimal partitioning of non-convex environments for minimum turn coverage planning," *arXiv preprint arXiv:2109.08185*, 2021.

[16] 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*. Springer, 2021, pp. 277–290.

[17] C. Hofner and G. Schmidt, "Path planning and guidance techniques for an autonomous mobile cleaning robot," *Robotics and autonomous systems*, vol. 14, no. 2-3, pp. 199–212, 1995.

[18] J. Palacin, T. Palleja, I. Valganón, R. Pernia, and J. Roca, "Measuring coverage performances of a floor cleaning mobile robot using a vision system," in *Proceedings of the 2005 IEEE international conference on robotics and automation*. IEEE, 2005, pp. 4236–4241.

[19] B. Yamauchi, "A frontier-based approach for autonomous exploration," in *Proceedings 1997 IEEE International Symposium on Computational Intelligence in Robotics and Automation CIRA'97.'Towards New Computational Principles for Robotics and Automation'*. IEEE, 1997, pp. 146–151.

[20] 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)*. IEEE, 2016, pp. 1237–1242.

[21] R. Bormann, F. Jordan, W. Li, J. Hampp, and M. Hägele, "Room segmentation: Survey, implementation, and analysis," in *2016 IEEE international conference on robotics and automation (ICRA)*. IEEE, 2016, pp. 1019–1026.

[22] J.-C. Latombe, *Robot motion planning*. Springer Science & Business Media, 2012, vol. 124.

[23] H. Choset, K. M. Lynch, S. Hutchinson, G. A. Kantor, and W. Burgard, *Principles of robot motion: theory, algorithms, and implementations*. MIT press, 2005.

[24] E. U. Acar, H. Choset, A. A. Rizzi, P. N. Atkar, and D. Hull, "Morse decompositions for coverage tasks," *The international journal of robotics research*, vol. 21, no. 4, pp. 331–344, 2002.

[25] 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, pp. 77–98, 2001.

[26] ——, "Spiral-stc: An on-line coverage algorithm of grid environments by a mobile robot," in *Proceedings 2002 IEEE International Conference on Robotics and Automation (Cat. No. 02CH37292)*, vol. 1. IEEE, 2002, pp. 954–960.

[27] R. Brooks, "A robust layered control system for a mobile robot," *IEEE journal on robotics and automation*, vol. 2, no. 1, pp. 14–23, 1986.

[28] D. W. Gage, "Randomized search strategies with imperfect sensors," in *Mobile Robots VIII*, vol. 2058. International Society for Optics and Photonics, 1994, pp. 270–279.

[29] E. Gat and G. Dorais, "Robot navigation by conditional sequencing," 1994.

[30] J. M. Palacios-Gasós, E. Montijano, C. Sagues, and S. Llorente, "Multi-robot persistent coverage using branch and bound," in *ACC*, 2016, pp. 5697–5702.

[31] T. Balch, "The case for randomized search," in *Workshop on Sensors and Motion, IEEE International Conference on Robotics and Automation, San Francisco, CA*, 2000, pp. 213–215.

[32] C. Tovey and S. Koenig, "Improved analysis of greedy mapping," in *Proceedings 2003 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2003)(Cat. No. 03CH37453)*, vol. 4. IEEE, 2003, pp. 3251–3257.

[33] J. Butzke and M. Likhachev, "Planning for multi-robot exploration with multiple objective utility functions," in *2011 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, 2011, pp. 3254–3259.

[34] J. Faigl and M. Kulich, "On determination of goal candidates in frontier-based multi-robot exploration," in *2013 European Conference on Mobile Robots*. IEEE, 2013, pp. 210–215.

[35] E. Vidal, N. Palomeras, K. Istenič, J. D. Hernández, and M. Carreras, "Two-dimensional frontier-based viewpoint generation for exploring and mapping underwater environments," *Sensors*, vol. 19, no. 6, p. 1460, 2019.

[36] B. Zhou, Y. Zhang, X. Chen, and S. Shen, "Fuel: Fast uav exploration using incremental frontier structure and hierarchical planning," *IEEE Robotics and Automation Letters*, vol. 6, no. 2, pp. 779–786, 2021.

[37] M. Pivtoraiko, R. A. Knepper, and A. Kelly, "Differentially constrained mobile robot motion planning in state lattices," *Journal of Field Robotics*, vol. 26, no. 3, pp. 308–333, 2009.

[38] M. Likhachev and D. Ferguson, "Planning long dynamically feasible maneuvers for autonomous vehicles," *The International Journal of Robotics Research*, vol. 28, no. 8, pp. 933–945, 2009.

[39] E. W. Dijkstra *et al.*, "A note on two problems in connexion with graphs," *Numerische mathematik*, vol. 1, no. 1, pp. 269–271, 1959.

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

[41] S. M. LaValle, *Planning algorithms*. Cambridge university press, 2006.