Coverage with Self-Induced Obstacles on Grids

Sarah Frenkel[1], David Parker[2], and Masoumeh Mansouri[1]

***Abstract*—Modelling environments as grids is a common approach in robotics, particularly for coverage path planning tasks, where the goal is to fully traverse an area or visit key points. In this paper, we introduce a new variant of coverage planning, inspired by applications such as open-pit mining, harvesting, and painting, where the robot's own actions modify the environment. Specifically, self-induced obstacles arise as a result of task execution, e.g. piles of rubble from drilling, and must be avoided in all future motion planning. We formally model these constraints by assuming that once a vertex is visited, it becomes non-traversable, and define an obstacle as self-induced if its existence depends on the set of previously visited vertices. This situation has not been addressed in existing formulations, which assume static obstacle placement. We provide a formal analysis of how the existence of solutions is affected by geometric constraints, such as vertex distances and robot turning radius, as defined by the Dubins vehicle. We also propose modelling the problem using self-deleting graphs based on the line graph of the grid. This provides a sufficient representation that captures the problem's dynamics and enables the use of general graph search algorithms. Our experimental evaluation demonstrates that our approach outperforms classical coverage path algorithms in terms of computation time and solution quality.**

***Index Terms*—Motion and Path Planning; Task and Motion Planning; Formal Methods in Robotics and Automation**

I. Introduction and Related Work

COVERAGE path planning is the problem of finding a path that fully traverses a given area. Current coverage planners can handle both static obstacles and dynamic obstacles, where the source of the dynamics is external to the robots, for example, a moving human [1]. In this paper, we consider a different kind of challenge: self-induced obstacles, which are created by the robot itself during operation, e.g. a pile of rubble generated by a drilling robot. Although these obstacles are not initially present, they appear in a somewhat predictable manner. For instance, consider the situation in an open-pit mine where a drilling operation creates a heap of rubble that obstructs the robot's movement later on. We refer to these obstacles, which arise in the environment as a result of the robot's previous actions, as *self-induced* obstacles. Self-induced obstacles are common in applications such as automated harvesting [2], [3], open-pit mining [4], [5], or lawn mowing [6]. An illustration of such an environment with self-induced obstacles is shown in Fig. 1.

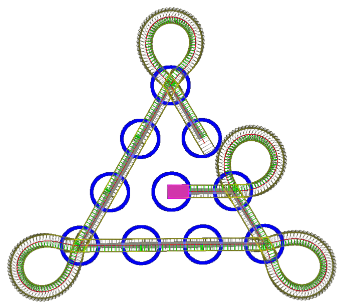


Fig. 1: Obstacles (blue) appear when the robot (pink) leaves their position. The illustrated path, starting at the centre, does not overlap with obstacles at previously visited positions.

The initial step in addressing coverage path planning typically involves decomposing the environment. Some approaches split the environment into several differently shaped areas [7], [8], others use a grid overlay to decompose the environment [9], [10]. The idea behind using decomposition is that each decomposed area, e.g. a grid cell, can be covered easily; hence, the challenge lies in determining the order in which the decomposed areas are visited. In this paper, we use a grid-based decomposition. This approach is particularly useful if the the motion pattern of a robot dictates some type of grid (e.g. in lattice-based planning [11]) or when the environment contains points of interest that are distributed in a grid-like fashion (e.g., in open-pit mining [5]). Grid-based coverage has been addressed using wavefront methods [9] or spanning tree strategies [10], but these methods do not consider the challenges posed by self-induced obstacles.

The problem of finding a path that visits every cell in a grid exactly once is the Hamiltonian path problem on grids. This problem has been extensively studied and, depending on the grid structure, is either polynomial-time solvable or NP-complete [12], [13]. However, these works assume a static environment and do not address obstacles that depend on the path itself. In this paper, we study the Hamiltonian path problem on grids with self-induced obstacles and with motion constraints: we approximate the robot as a *Dubins vehicle*, which can only move forward and has a minimal turning radius [14]. Hence, a turn may require space that overlaps with a previously created obstacle, making the move impossible. Fig. 2 illustrates this: the first path is invalid as it collides with an earlier created obstacle.

Classical coverage-search methods operate on a fixed adjacency. In our setting, every visited vertex creates a self-induced obstacle whose geometry interacts with the motion constraints, so the set of feasible successors depends on the entire path history. Thus, classical methods cannot ensure that a chosen

Manuscript received: September 23, 2025; Revised December 19, 2025; Accepted January 14, 2026.

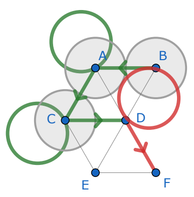
This paper was recommended for publication by Editor Lucia Pallottino upon evaluation of the Associate Editor and Reviewers' comments.

[1] School of Computer Science, University of Birmingham, United Kingdom `sxc1431@student.bham.ac.uk`, `m.mansouri@bham.ac.uk`

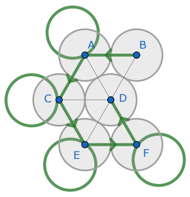
[2] Department of Computer Science, University of Oxford, United Kingdom

Masoumeh Mansouri is UK participant in Horizon Europe Project CONVINCE, and supported by UKRI grant number 10042096. For the purpose of open access, the authors have applied a Creative Commons Attribution (CC BY) license to any Accepted Manuscript version arising.

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



(a) Invalid path.



(b) Valid path.

Fig. 2: Two different paths through a grid, starting from vertex B. The path in (a) is invalid, since it collides with the obstacle (in grey) created at vertex B.

next vertex will remain reachable, and these history-dependent reachability changes place the problem outside the scope of existing coverage techniques.

Problems involving dynamic environments have been approached through frameworks like dynamic graphs [15], [16] or the Canadian Traveller Problem [17], where edges are blocked or available based on exogenous conditions. In contrast, our model involves obstacle creation, where the robot's own traversal determines future obstructions.

A naive idea is to replan the path after each step, adapting to the newly created obstacles. However, such local strategies often fail to anticipate how early decisions constrain future movement, leading to dead ends. Even choosing the wrong start-vertex can lead to failure. This highlights the need to consider interactions between motion primitives and obstacle creation at an early stage of the planning.

To address this, we build on the framework of self-deleting graphs [18], where the traversal of a vertex can delete edges. In our model, we interpret the motion constraints of the Dubins vehicle and the placement of self-induced obstacles as a special case of self-deleting graphs. Earlier work has used a random function [18] or circular obstacles [19] that delete edges. Our model uses geometry-informed deletion rules based on motion primitves and obstacle placement.

Contributions. We formally define the Hamiltonian path problem on grids with self-induced obstacles for Dubins vehicles and study how geometric parameters such as turning radius, grid spacing, and obstacle size influence solveability. We evaluate classical coverage strategies and highlight their limitations in this setting. We then propose a novel graph-theoretic formulation using self-deleting line graphs, enabling general search algorithms. We demonstrate the applicability of this model through a depth-first search approach which guarantees that an offline solution will be found whenever one exists and removes the need for any online replanning.

II. Preliminaries

In this section, we introduce the terms that will be used throughout the paper. A *graph G* is a simple undirected graph consisting of vertices $V(G)$ and edges $E(G)$. The *line graph* of G has a vertex for every edge in G and two vertices in the line graph are connected if the corresponding edges in G share a vertex. A *complete graph* on n vertices K_n is the graph with n vertices, where every pair of distinct vertices is connected by an edge.

Let a *triangular grid graph T* be an embedded graph, where every face, except the outer face, is an equilateral triangle. The *shape of a grid T* refers to the shape of the outer face of T.

To account for the motion constraints of a robot we use the model of a *Dubins vehicle*. This is a vehicle that can only move forward and has a fixed turning radius. The optimal path for a Dubins vehicle given a start and end-position and direction has been widely studied [14], [20], [21]. In this paper we assume any change in the direction of the vehicle happens at a vertex, thus start and end point are identical, only the direction the vehicle changes. An optimal Dubins path which results in a left-turn with identical start and end point is described by a LRL-movement, that is a left-turning circle arc, followed by a right-turning circle arc and finished with another left-turning circle arc. A right turn can be obtained by mirroring the movements. We pre-compute the optimal paths for a Dubins vehicle for every turn, that a robot would have to make to traverse the grid. This gives a set of two motion primitives; two examples of which are shown in Fig. 3b and 3c.

III. Problem description

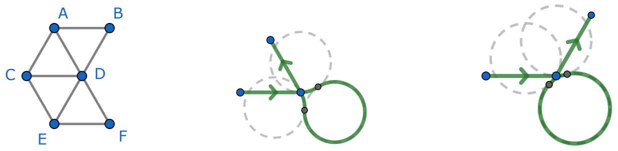
In this section, we formally describe the central problem addressed in this paper. We limit the analysis of this paper to the movement primitives of a Dubins vehicle. Alternative motion primitives could be obtained by sampling the motions of an actual vehicle or by applying different theoretical approximations of movements, e.g. Reeds-Shepp [21]. We finish this section with some theoretical observations and explain interesting cases.

Problem 1. *Given a grid graph G on n vertices, a turning radius r and a radius o, the* Coverage Path problem of a Dubins vehicle on a Grid with Obstacle Creation *(CP-DGO), described by the the tuple (G, r, o), is to find a path $P = (p_1, ..., p_n)$, where the p_i are vertices of G such that:*

- *every vertex of G is visited exactly once;*
- *the movement between two adjacent vertices p_i and p_{i+1} on P consists of an optimal turning Dubins's path with turning radius r, and a straight line motion from p_i and p_{i+1}; and*
- *the movement between every pair of adjacent vertices p_i and p_{i+1} on P does not intersect the obstacle at any previously visited vertex p_j with $j < i$, given by a disk of radius o centered at p_j.*

We will use the following as a running example throughout the paper.

Example 1. *We now describe an instance (G, r, o) of CP-DGO. Let G be a triangular grid graph on six vertices $A, B, ..., F$, as shown in Fig. 3a. Two adjacent vertices have distance 1. The vehicle used to traverse the grid is a Dubins vehicle with a fixed turning radius of 1. To traverse the grid, the vehicle will make two types of turns at vertices, $2\pi/3$-turn and a $\pi/3$-turn. The pre-computed set of motion primitives M is shown in Fig. 3b and 3c. After visiting a vertex v, an obstacle with radius $o = 0.5$ is created.*



(a) (b) $\frac{2\pi}{3}$-turn (c) $\frac{\pi}{3}$-turn

Fig. 3: The grid G and the motion primitives M of Example 1

Solving Problem 1 on (G, r, o) is to find a path visiting all vertices in the grid, that does not collide with any obstacles. A valid path and an invalid path are shown in Fig. 2(b).

Given an instance of Problem 1 we analyse the relation between vertex distance, turning radius of the Dubins vehicle and size of the created obstacle for triangular grids. This gives us insight into which setups do not cause obstacles that obstruct the motion primitives. We then prove which shapes of grids can be solved, even if turns are obstructed.

A. The relation between grid, turning radius and obstacle size in a triangular grid

In this section, we geometrically analyse how turning radius, grid spacing and obstacle size affect collisions between motion primitives and obstacles. If the creation of obstacles at the vertices does not obstruct later movements, the problem of finding a path through the grid reduces to the classic Hamiltonian cycle problem on grids. For results on Hamiltonicity of grids we refer to [13].

We remind the reader that, throughout this paper, we assume that a turning action does not happen on the path between two vertices but during the visit to a vertex. This turning action is given through the motion primitives. We compute the motion primitives by finding optimal paths for a Dubins vehicle.

Lemma 1. *Let r be the turning radius of a Dubins vehicle, d the distances between two adjacent vertices in a triangular grid, and o the radius of the obstacles, that centers at a vertex. Let further $r \leq d$, $o \leq d/2$ and a constant $f = \sqrt{(8-\sqrt{3})/2} - 1/2$. Then the obstacles do not obstruct the turning movement at a vertex if the following two inequalities hold:*

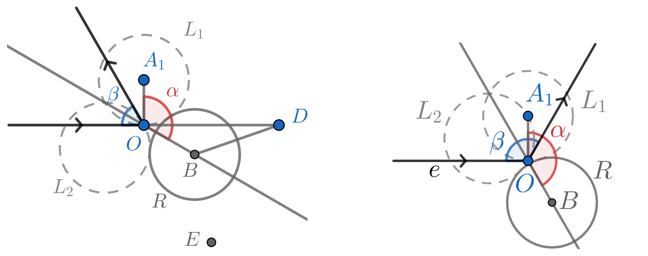
$$d - o > \left(1 + \sqrt{\frac{7}{2} - \frac{\sqrt{3}}{2}}\right) r,$$

$$(r+o)^2 > d^2 + (rf)^2 - \frac{\sqrt{3}}{2} rfd$$

Proof. We analyse left-turns of a Dubins vehicle; right-turns are symmetric. At a vertex O the vehicle performs an Left–Right–Left (LRL) path, consisting of three arcs of radius r, as seen in Fig. 4b. The first and last arcs lie on circles L_1, L_2 tangent to the incoming and outgoing edges. The middle arc lies on a circle R of radius r, tangent to both L_1 and L_2.

The center B of R lies at distance $2r$ from both A_1 and A_2. Let β be the turning angle and $\alpha = \frac{\pi-\beta}{2}$ the angle between A_1O and OB. Using the law of cosines we obtain for the $\frac{\pi}{3}$-turn:

$$|OB| = -\frac{\sqrt{3}}{2} r \pm r\sqrt{4 - \frac{1}{2}} = \left(\sqrt{\frac{7}{2}} - \frac{\sqrt{3}}{2}\right) r$$



(a) A deconstructed $\frac{2\pi}{3}$-turn. (b) A deconstructed $\frac{\pi}{3}$-turn.

Fig. 4: Deconstructed LRL-turn at vertex O, with variable names as used in the proof. The vehicle approaches from the left and leaves the vertex towards the upper right or left.

Let d be the distance from O to a neighbouring vertex C and o the obstacle radius. A circular obstacle at C does not intersect the motion primitive if

$$d - o > \left(1 + \sqrt{\frac{7}{2} - \frac{\sqrt{3}}{2}}\right) r. \tag{1}$$

For the $\frac{2\pi}{3}$-turn (Fig. 4a), we obtain

$$|OB| = -\frac{3}{2} r \pm r\sqrt{4 - \frac{\sqrt{3}}{2}} = \left(\sqrt{\frac{8-\sqrt{3}}{2}} - \frac{1}{2}\right) r$$

We set $f = \sqrt{\frac{8-\sqrt{3}}{2}} - \frac{1}{2}$.

Let D be the nearest vertex to circle R. Due to the law of cosine we get $|BD|^2 = d^2 + rf^2 - rfd\cos\frac{\pi}{6}$ and for the circle R to not be intersect by an obstacle around D we get $|BD| > r + o$. Hence,

$$(r+o)^2 < d^2 + rf^2 - \frac{\sqrt{3}}{2} rfd \tag{2}$$

If the inequalities (1) and (2) hold, a circular obstacle of radius r at a vertex does not obstruct any turning motion. □

The previous lemma helps us to characterise instances of the CP-DGO which are solvable with traditional approaches.

Corollary 2. *The problem of solving Problem 1 on (G, r, o) reduces to the classic Hamiltonian cycle problem on grids,if the grid G is triangular and the inequalities of Lemma 1 hold.*

B. Analysing different grid shapes

In this section, we investigate the grid shapes that allow Problem 1 to be solvable, even when the geometric constraints outlined in Lemma 1 are not satisfied.

Lemma 3. *Let G be a triangular grid in the shape of a triangle. Then we can solve Problem 1 over (G, r, o).*

Proof. We proceed by induction on the side length n of the triangular grid T_n. For $n = 1$, the path consists of the single vertex, which is a corner.

Assume T_n admits a covering path ending in a corner. For T_{n+1}, remove $n+1$ boundary vertices to obtain T_n and apply the induction hypothesis. Adding the boundary back (Fig. 5), the path ends at a boundary vertex v of T_n. From v the path can be extended to a neighbour u, where a turning move (outside the triangle) allows us to proceed along the new boundary. This extension covers all remaining vertices, ending at a corner of T_{n+1}. □

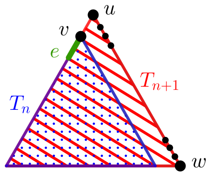


Fig. 5: How to extend a path covering a triangle T_n with side length n to cover a triangle T_{n+1} with side length $n+1$.

Remark 4. *We can cover any triangular grid shaped like a regular hexagon or a parallelogram in a corresponding way.*

C. Fixed obstacles

Real-world environments can have fixed obstacles, for example, a table in a room or a boulder in an outdoor setting. Environments with fixed obstacles can still be approximated with a grid graph. To account for the obstacles we delete those vertices from the grid that coincide with the obstacles, this results in a "hole" in the grid. A *hole* in a triangular grid is an inner face that is not a small triangle.

Lemma 5. *If a triangular grid graph G is in the shape of a triangle, a regular hexagon or a parallelogram and it contains a single central hole in the same shape as the grid we can solve Problem 1 over (G, r, o).*

Sketch.. Analogue to the proof of Lemma 3. □

Our proofs do not extend to multiple holes or to holes that are not centrally positioned. In fact, our experiments in Section VII suggest that complete coverage is only possible when a single obstacle lies centrally. In the remaining cases our solver provides a maximal coverage path.

IV. Modelling using self-deleting graphs

In the previous sections, we discussed cases where the CP-DGO is "easy" to solve. Corollary 2 characterises cases of CP-DGO that reduce to the classical Hamiltonian Path Problem (HPP) on grids. In these cases, we can use algorithms that solve the HPP on grids to solve Problem 1. The HPP on grids is hard in general but there are several grid-types that have polynomial time algorithms [13]. The proofs presented in Section III-B indicate procedures with which we can quickly solve the CP-DGO when the underlying triangular grid is in the shape of a triangle, a hexagon or a parallelogram. Since the number of shapes that can be solved "easily" is highly limited, we need a more general approach.

In this section, we introduce a novel way of modelling Problem 1 that is general and not limited to the above cases. In particular, we propose using self-deleting graphs [18]. This graph class allows us to model the dependencies between visits to vertices and the resulting changes in the traversability of the grid. In the following section, we describe in detail how the modelling is done. We start by reiterating the definition of self-deleting graphs and then explain how to model the CP-DGO with self-deleting graphs.

Definition 1. *[18] A* self-deleting graph *S is a tuple $S = (G, f)$ where $G = (V, E)$ is a simple, undirected graph and $f : V \to 2^E$. The function f specifies for every vertex $v \in V$ which edges $f(v)$ are deleted from E if the vertex v is processed. We refer to f as the* delete-function.

If a vertex v is *processed*, we delete edges $f(v)$ from G. For a self-deleting graph S and set $X \subseteq V$ of vertices, the *residual graph G_X* of S after processing X is defined as:

$$G_X = G \setminus \bigcup_{v \in X} f(v).$$

We call a simple path $p = (v_1, ..., v_x)$ in a self-deleting graph *f-conforming* if for every $1 \leq i < x$ the edge $e_i = \{v_i, v_{i+1}\}$ is in the residual graph $G_{\{v_1,...,v_i\}}$. An f-conforming simple path p traverses the graph G while processing every vertex on p when it is visited.

To model CP-DGO through a self-deleting graph $S = (L(G), f)$ we define the underlying graph $L(G)$ and a delete-function f as follows.

a) The underlying graph $L(G)$: To model how the traversability of the grid changes as the path gets longer, we use the line graph $L(G)$ of the grid as the underlying graph of a self-deleting graph. Every possible transition at a vertex in the original graph is represented by an edge in the line graph. The line graph of a grid graph consists of a K_d for every vertex v in the original grid, where d is the degree of vertex v, and two K_d share a vertex if the vertices they represent are connected by an edge in the original grid. The process is shown in Fig. 6. Here, vertex A in the original grid is represented by the K_3 consisting of AB, AD, AC in the line graph.

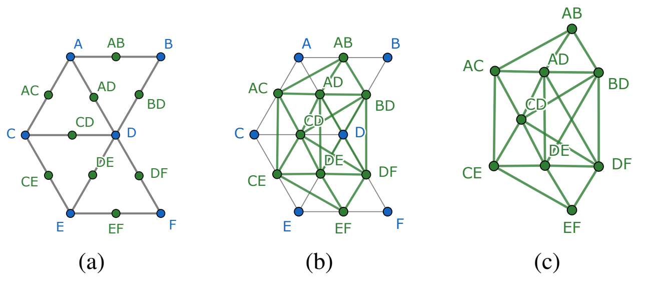


Fig. 6: Constructing the line graph of the grid in Example 1: (a) add vertices for each edge, (b) connect those sharing a vertex, (c) final line graph.

b) The delete-function f: A vertex xy in the line graph represents an edge xy of the grid. When the path visits xy, edges are deleted for two reasons:

1) *Strict deletes:* to ensure each grid vertex is visited at most once, all edges in the K_ds of x or y not incident with xy are removed.
2) *Motion deletes:* to model how visiting one vertex blocks transitions at others, single edges are removed depending on the obstacle and its overlap with motion primitives.

Observation 1. *If a CP-DGO instance satisfies the ratios of Lemma 1, the corresponding SDG has no motion deletes.*

Observation 2. *Consider a CP-DGO with grid distance d, turning radius $r < 0.5d$, and obstacle size violating Lemma 1.*

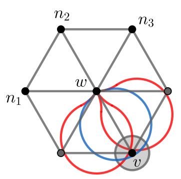


Fig. 7: Turns at vertex w obstructed by an obstacle at v. The edges from n_1, n_2, n_3 to w form the set N_w. The blue curve describes the Dubins path performing the $\pi/3$-turn at w, coming from n_1 to n_3. The red curves describe the two $\pi/3$-turns.

Then the SDG S contains motion deletes: an obstacle at vertex v can obstruct up to three turns at each of its (up to six) neighbours, as shown in Fig. 7.

Let N_w be the edges incident to w but not to other neighbours of v. The obstacle at v blocks every turn at w that transitions between edges of N_w, so f deletes all of these combinations.

To solve a CP-DGO instance, we construct the corresponding self-deleting graph and search for an f-conforming path of length n, the number of grid vertices.

Example 1. ***(continued)*** *In Fig. 2a we see, that after visiting vertex B the turn at vertex D is obstructed. We can model this by adding the edge between CD and DF in the line graph to the delete-function of any vertex of the line graph that represents the vertex B of the original grid.*

Lemma 6. *Let $D = (G, r, o)$ be an instance of a CP-DGO and $S = (line(G), f)$ the corresponding self-deleting graph to D. Then an f-conforming path through S is equivalent to a valid path through D.*

Sketch.. The proof results from the construction of the delete-function. □

Corollary 7. *A path of length n through S corresponds to a coverage path through D.*

A. How to model fixed obstacles

We can model a fixed obstacle in the environment by preprocessing the vertices which correspond to the position of the obstacle. So, if we have grid G and a fixed obstacle is present at a vertex v, we delete all edges from the line graph of G that would be deleted by the motion deletes of any vertex w in the line graph, that corresponds to an edge incident to v. Since the final path will not visit the vertex v we also delete all vertices of the line graph, that correspond to edges incident to vertex v.

Example 2. *We add to Example 1 a fixed obstacle at vertex C. The vertex C obstructs the transition from B to F via D. So we remove the edge BD and DF from the line graph. Additionally, we remove all vertices from the line graph that correspond to edges incident to C: AC, CE, CD.*

V. Solving the CP-DGO

In this section, we introduce different approaches for solving the CP-DGO. We consider three classical approaches to coverage path planning: two grid-based methods, the wavefront algorithm and the path transform algorithm [9], and a hybrid planner based on the Boustrophedon method. Additionally, we use a depth-first search (DFS) approach on the corresponding self-deleting graph of a CP-DGO. We describe and analyse each algorithm and end this section with an experimental comparison.

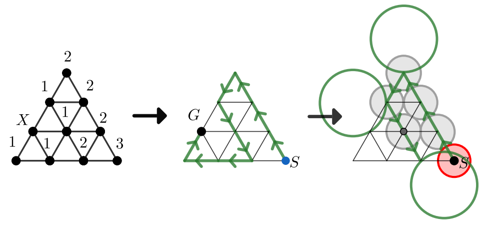


Fig. 8: Run of the wavefront algorithm. Vertices are labelled with the distance to the goal vertex X. The start vertex S is furthest away. The middle figure depicts the visiting order of the vertices. The last figure shows the Dubins path, highlighting the obstruction at the first vertex.

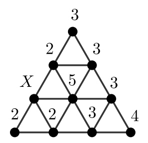
A. Using classic coverage path planners

For grids, several coverage path planners have been developed [9], [22]. In this section, we focus on the wavefront algorithm and the path-transform algorithm, the latter being an extension of the former. We evaluate them on CP-DGO instances by comparing feasibility of the generated paths and their computation times, and conclude with a discussion of the boustrophedon method.

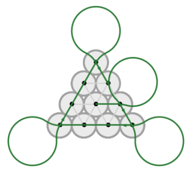
The *wavefront* algorithm was first described in [22]. While the original algorithm was applied to a square grid, the core principle of the algorithm, assigning each vertex a distance to the goal vertex, works on various grid types. During the initial step of the wavefront algorithm, all vertices are labelled with their distance to the goal vertex. The coverage path is determined by selecting a vertex with the highest distance as the starting point and then iteratively progressing along neighbouring vertices with the highest distance until reaching the goal vertex. Fig. 8 shows a coverage path through a triangular grid computed by the wavefront algorithm.

In Section III-B we have seen that a path that follows the shape of an environment can lead to a valid coverage path. However, the wavefront algorithm produces a path with a significant number of turns [9]. To counteract this, the *path transform* algorithm adds a second property to every vertex: it calculates the shortest distance to any obstacle. A path transform value for each vertex is then calculated as a function of the obstacle distance and the paths to the goal vertex. The coverage path is then created the same way as in the Wavefront algorithm, beginning at the vertex with the highest path transform value. An result of the path transform algorithm is shown in Fig. 9.

These algorithms were designed for grids without self-induced obstacles. We adopt them for the CP-DGO as shown in Algorithm 1. We select a random goal vertex and calculate the vertex labels. Based on this, the algorithms generate a path. To make the algorithm applicable to the CP-DGO, the



(a) Initial step: $pt(v)$ is the distance of v to X plus the square of the distance of v to the boundary.



(b) Path found: start vertex in the center with max pt. Dubins path (green) avoids obstacles.

Fig. 9: Run of the path transform algorithm.

path is then checked for validity. If it fails to cover the environment a new goal vertex is chosen and the vertex labels are recomputed. Algorithm 1 shows the path transform algorithm. The wavefront algorithm differs in line 14. The wavefront chooses the next vertex on the basis of $d(v)$.

Algorithm 1 Path transform algorithm for CP-DGO

```
Input: instance of CP-DGO $(G, r, o)$;
Output: Coverage path $p$ of $G$
 1: function PATH-TRANSFORM($G$)
 2:     $obsDist(v) \leftarrow$ distance from $v$ to outer face
 3:     $goals \leftarrow$ shuffled stack of $V(G)$
 4:     while $p$ not valid coverage path and $goals \neq \varnothing$ do
 5:         $p \leftarrow$ empty path; $goal \leftarrow goals.pop$
 6:         $d(v) \leftarrow dist(v, goal)\ \forall\ v \in G$ using a BFS
 7:         $pt(v) \leftarrow d(v) + obsDist(v)\ \forall\ v \in G$
 8:         $s \leftarrow$ vertex where $d(s)$ is maximal; $p \leftarrow \{a\}$
 9:         while $|p| < |V(G)|$ do
10:             Append to $p$ the neighbour of $p.last$ with maximal
                $pt$, not yet on $p$
11:         end while
12:     end while
13:     return $p$
14: end function
```

Experiments showed that picking a "good" goal vertex reduces runtime. In particular, choosing a vertex in the middle of an outer side speeds up finding a solution. We use this observation as a heuristic and perform a second run of the path transform algorithm on the dataset with pre-set goal vertices.

B. Hybrid planner

The hybrid planner first computes a global coverage order of all grid vertices using a back-and-forth sweep, as used in boustrophedon coverage. For each consecutive pair in this order, a local planner computes the shortest Dubins trajectory. This represents a global-local approach where a high-level sweep pattern is combined with a continuous local motion planner [1].

C. DFS on the corresponding self-deleting graph

A direct DFS over the CP-DGO is not feasible because each visited vertex creates an obstacle. This changes the successors in the DFS after every step and the search cannot predict which moves remain reachable. To overcome this lack of predictability, we propose to use the corresponding self-deleting graph to solve the CP-DGO.

In Section IV, we described how self-deleting graphs can be used to model a CP-DGO. To find a solution for a CP-DGO instance, we run a DFS on the corresponding self-deleting graph, as previously used in [18]. We stop the DFS if a complete path is found. If no complete path can be found, the algorithm returns the longest possible path.

The proof of Lemma 3 uses a spiralling coverage path. We can use this observation and start the DFS at a central vertex of the grid. In Section VI we will compare this informed approach - starting from a central vertex - with an uninformed approach that randomly selects a start vertex.

VI. Experiments

In this section, we evaluate the algorithms introduced in Section V using a synthetic dataset. All experiments were performed on a Macbook Air with an Apple M1 CPU (3.2 Ghz, 8 cores) and 8GB RAM.

A. Dataset

The dataset consists of 44 triangular grids in the shape of triangles and parallelograms with between 6 and 40 vertices. The distance between two vertices is $d = 1$. We fix the set of motion primitives M to the optimal path of a Dubins vehicle with radius d, as pictured in Fig. 3b and 3c. The size of the obstacles is $0.5d$. These parameters are consistent with the running example presented in Example 1. Lemma 1 implies that in an instance with these parameters the self-induced obstacles do obstruct turns. Due to Lemma 3 and Remark 4 all of these instances are solvable.

B. Results

The wavefront algorithm produces a path that contains many turns. As turns are the critical movement for the Dubins vehicle is not surprising, that this approach does not lead to a single viable path in our experiments.

The hybrid planner using a global planner for the vertex ordering and a local planner to create a Dubins trajectory also fails to find a solution to a CP-DGO. Since the global planner fixes an order that does not account for future self-induced obstacles, the local Dubins step is often asked to reach a vertex that has already become inaccessible. Fig. 10 illustrates two such situations, where every locally computed Dubins path to the next vertex is blocked by the newly created obstacles.

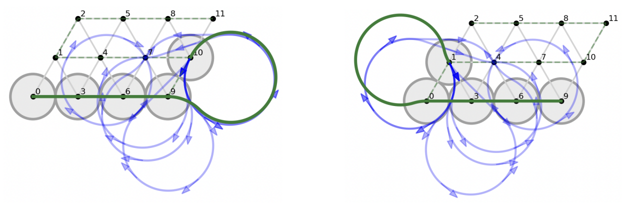


Fig. 10: Two failure scenarios of the hybrid solver. The green curve shows the trajectory, while the blue curves are the locally computed Dubins paths toward the next vertex. All of them are blocked by the newly created obstacles (grey).

Lemma 3 and Remark 4 prove that each problem instance within the dataset is solvable, so there always exists a valid

TABLE I: Computing time (s) of path transform vs. DFS solver for CP-DGO on triangles (top) and parallelograms (bottom). Runs exceeding 20 min were aborted (TO). The hybrid solver failed to solve a single instance of the dataset.

		DFS		Path Transform		Hybrid
Shape	Size	Uninf.	Inf.	Uninf.	Inf.	
Triangle	6	0.00	0.00	0.00	0.00	-
	15	0.15	0.04	0.19	0.03	-
	21	3.3	0.90	44.62	2.6	-
	28	116.82	26.63	TO	TO	-
	36	TO	795.1	TO	TO	-
Parallelogram	9	0.00	0.00	0.00	0.00	-
	16	0.25	0.08	0.32	0.08	-
	20	1.90	0.15	8.50	2.05	-
	25	23.32	0.23	884.4	107.3	-
	30	138.68	2.71	TO	TO	-
	35	2976.25	129.0	TO	TO	-
	40	TO	1086	TO	TO	-

coverage path. The path transform algorithm and the DFS-approach both succeed in finding a coverage path for every problem instance. We ran both algorithms twice on each grid, the first run being an uninformed search and the second an informed search, as outlined in the relevant sections, the results are displayed in Table I.

Using an informed search, either with DFS or the path transform algorithm, significantly improves computing time for both parallelogram- and triangle-shaped grids. Despite this, the computing time still grows exponentially, as illustrated in Figures 11a and 11b. The informed DFS on corresponding the self-deleting graph consistently outperforms the path transform method. Both approaches restart if a found path is not valid, with a changed start or target vertex. However, the DFS has an advantage here: its self-deleting graph needs to be computed only once and can be reused across attempts. In contrast, the path transform method needs to recalculate the vertex labels for each new run, resulting in considerable computational overhead.

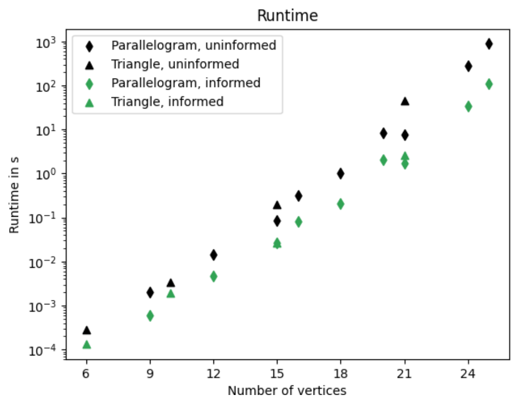
VII. Statistical analysis of of the CP-DGO with fixed obstacles

In Section III-C, we analysed the impact of fixed obstacles on the solvability of a CP-DGO from a theoretical point of view. In Lemma 5, we showed that full coverage is achievable for small, centrally located obstacles. In this section, we extend that analysis with a statistical evaluation of CP-DGO instances containing one or two fixed obstacles.

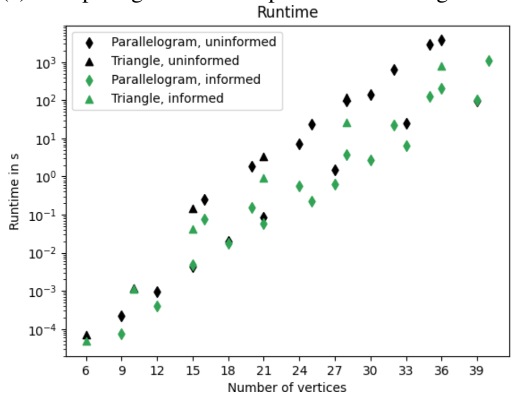
To do this, we created a dataset by randomly placing one or two obstacles on triangular grids shaped as parallelograms with dimensions 5×5 and 5×6. Due to the small size, we exhaustively explored all 1525 possible positions of blocked vertex. For each instance, we measured the resulting path length and computation time, as summarised in Table II.

Our analysis reveals that with one obstacle, full coverage is possible only if it lies centrally, indicating that Lemma 5 is tight. Hence, only approximately 4% or 7% of the total dataset have a complete coverage path. If the obstacle is not in the center, complete coverage cannot be achieved.

Introducing a second fixed obstacle further reduces the solvability. Full coverage is again only possible when both



(a) Computing time of the path transform algorithm.



(b) Computing time of the DFS approach.

Fig. 11: Computing time comparison for different shaped grids using different algorithms. Priming the algorithm with "good" goal vertices (green) outperforms the uninformed search (black) in both cases.

obstacles are placed adjacently in the center of the grid. Otherwise, a complete coverage path cannot be found.

TABLE II: Comparison of average runtime (rt) and maximum path lengths in different grids depending on the number of fixed obstacles.

Grid	Path length	0 fixed obs. rt (s)	1 fixed obs. rt (s)	1 fixed obs. % of cases	2 fixed obs. rt (s)	2 fixed obs. % of cases
5 × 5	25	23.32	–	–	–	–
	24	–	0.11	4%	–	–
	23	–	3.99	80%	0.98	1.3%
	22–19	–	4.51	16%	0.57	80%
	16–18	–	–	–	0.20	18.6%
5 × 6	30	138.68	–	–	–	–
	29	–	6.97	7%	–	–
	28	–	44.22	80%	0.79	0.2%
	24-27	–	58.95	13%	2.63	76.8%
	21–23	–	–	–	0.84	23%

VIII. Discussion and future work

We introduced a new coverage path problem on grids with obstacle creation. We performed a theoretical analysis on geometric constraints and solvability of the CP-DGO. Based on modelling the CP-DGO using self-deleting graphs we suggest a solver. This solver outperforms classic coverage path planners. However, neither solver scales to larger environments with more than 40 vertices. The analysis of the

CP-DGO with fixed obstacles reveals that optimal solutions are only achievable in highly specific instances, highlighting the inherent difficulty of the CP-DGO.

In our paper, we introduced several limitations: We only studied triangular grids, one motion type and stipulated that the vehicle can only turn at vertices, not while traversing edges. In the following we justify these limitations.

a) Types of grid: We work with triangular grids. Square grids are common in discretisations but restrict movement to four directions, which oversimplifies real environments. Triangular grids offers six possible directions. By contrast, adding diagonals to square grids increases flexibility but introduces unequal edge lengths, turning coverage into a shortest-path problem.

In a CP-DGO with large obstacles, movements that deviate from the six grid directions almost always intersect the previously created obstacles. Edges at different angles are removed immediately because their straight-line segments would collide with obstacles. Hence, restricting headings to the six grid directions does not remove feasible solutions; it removes only those edges that are already blocked. Fig. 12 illustrates how previously formed obstacles obstruct straight edges in additional directions.

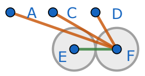


Fig. 12: Additional edges in a triangular grid pass through self-induced obstacles.

b) Locations of turns: Restricting turns to vertices avoids combinatorial explosion: the self-deleting graph of a CP-DGO with n vertices then has $3n$ vertices and $15n$ edges. Allowing turns between vertices yields 36 transition types (by entry and exit angles), and the approach to one vertex restricts options at the next. The approaches from this paper would not be able to adequately model these constraints. Future work should investigate partial solutions and heuristics that can handle this complexity. Mid-edge turns are in principle possible for a Dubins vehicle, but their feasibility depends on the exact obstacle positions. As obstacles emerge, the space needed to perform a turn along an edge may or may not be available, making such manoeuvres unreliable. Constraining turns to vertices ensures that all motion primitives remain consistently feasible throughout coverage. Where we relaxed this requirement for the hybrid planner (Sec. V-B), no feasible solution was found.

c) Movement constraints: We used a Dubins vehicle as a running example. However, this can be replaced by another type of motion constraint vehicle, e.g. Reeds-Shepp [23]. Using different motion primitives will lead to a different delete-function in the corresponding self-deleting graph. The algorithm based on the self-deleting graph will remain the same. To what extent the resulting paths of other algorithms are still usable is not clear.

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] S. Scheuren, S. Stiene, R. Hartanto, J. Hertzberg, and M. Reinecke, "Spatio-temporally constrained planning for cooperative vehicles in a harvesting scenario," *KI-Künstliche Intelligenz*, vol. 27, no. 4, pp. 341–346, 2013.

[3] A. Ullrich, J. Hertzberg, and S. Stiene, "Ros-based path planning and machine control for an autonomous sugar beet harvester," in *Proceedings of International Conference on Machine Control & Guidance, (MCG-2014)*, 2014.

[4] M. Mansouri, H. Andreasson, and F. Pecora, "Hybrid reasoning for multi-robot drill planning in open-pit mines," *Acta Polytechnica*, vol. 56, no. 1, pp. 47–56, 2016.

[5] M. Mansouri, F. Lagriffoul, and F. Pecora, "Multi vehicle routing with nonholonomic constraints and dense dynamic obstacles," in *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2017, pp. 3522–3529.

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

[7] H. Choset and P. Pignon, "Coverage path planning: The boustrophedon cellular decomposition," in *Field and service robotics*. Springer, 1998, pp. 203–209.

[8] 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.

[9] A. Zelinsky, R. A. Jarvis, J. Byrne, S. Yuta, *et al.*, "Planning paths of complete coverage of an unstructured environment by a mobile robot," in *Proceedings of international conference on advanced robotics*, vol. 13. Citeseer, 1993, pp. 533–538.

[10] 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.

[11] A. Botros and S. L. Smith, "Computing a minimal set of t-spanning motion primitives for lattice planners," in *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2019, pp. 2328–2335.

[12] E. M. Arkin, S. P. Fekete, K. Islam, H. Meijer, J. S. Mitchell, Y. Núñez-Rodríguez, V. Polishchuk, D. Rappaport, and H. Xiao, "Not being (super) thin or solid is hard: A study of grid Hamiltonicity," *Computational Geometry*, vol. 42, no. 6-7, pp. 582–605, 2009.

[13] E. D. Demaine and M. Rudoy, "Hamiltonicity is hard in thin or polygonal grid graphs, but easy in thin polygonal grid graphs," *arXiv preprint arXiv:1706.10046*, 2017.

[14] L. E. Dubins, "On curves of minimal length with a constraint on average curvature, and with prescribed initial and terminal positions and tangents," *American Journal of mathematics*, vol. 79, no. 3, pp. 497–516, 1957.

[15] E. Aaron, D. Krizanc, and E. Meyerson, "Multi-robot foremost coverage of time-varying graphs," in *International Symposium on Algorithms and Experiments for Sensor Systems, Wireless Networks and Distributed Robotics*. Springer, 2014, pp. 22–38.

[16] O. Michail and P. G. Spirakis, "Traveling salesman problems in temporal graphs," *Theoretical Computer Science*, vol. 634, pp. 1–23, 2016.

[17] C.-S. Liao and Y. Huang, "The covering Canadian traveller problem," *Theoretical Computer Science*, vol. 530, pp. 80–88, 2014.

[18] S. Carmesin, D. Woller, D. Parker, M. Kulich, and M. Mansouri, "The Hamiltonian cycle and travelling salesperson problems with traversal-dependent edge deletion," *Journal of Computational Science*, vol. 74, p. 102156, 2023.

[19] M. Kulich, D. Woller, S. Carmesin, M. Mansouri, and L. Přeučil, "Where to place a pile?" in *2023 European Conference on Mobile Robots (ECMR)*. IEEE, 2023, pp. 1–7.

[20] J.-D. Boissonnat, A. Cérézo, and J. Leblond, "Shortest paths of bounded curvature in the plane," *Journal of Intelligent and Robotic Systems*, vol. 11, pp. 5–20, 1994.

[21] H. J. Sussmann and G. Tang, "Shortest paths for the Reeds-Shepp car: a worked out example of the use of geometric techniques in nonlinear optimal control," *Rutgers Center for Systems and Control Technical Report*, vol. 10, pp. 1–71, 1991.

[22] R. Jarvis and J. Byrne, "Robot navigation: Touching, seeing and knowing," in *Proceedings of the 1st Australian conference on artificial intelligence*, vol. 69, 1986.

[23] J. Reeds and L. Shepp, "Optimal paths for a car that goes both forwards and backwards," *Pacific journal of mathematics*, vol. 145, no. 2, pp. 367–393, 1990.