Aerial Coverage Planning for Areas Hidden from the View of a Moving Ground Vehicle

Barry Gilhuly and Stephen L. Smith

***Abstract*— We consider the problem of path planning for a UAV, deployed to provide sensor coverage ahead of a moving ground vehicle. The ground vehicle travels a fixed route through an uncertain environment and requires information about the area ahead. Given this route, the UAV planner calculates the regions to be covered and the time by which each must be covered, as an Orienteering Problem with Time Windows (OPTW) and solves it using a Mixed Integer Linear Program (MILP). To improve scalability, we prove that the optimization can be partitioned into a set of smaller problems, each of which may be solved independently without loss of overall solution optimality. Finally, we demonstrate a method of limited loss partitioning, which can perform a trade-off between improved solution time and a bounded objective loss. All of our results are validated in simulation.**

I. Introduction

In applications such as search and rescue, disaster recovery, or transportation of goods and personnel in hostile environments, a ground vehicle may have to traverse uncertain and potentially dangerous terrain. Existing maps may be used for route planning, but may differ from the current conditions. Further, obstructions such as buildings may conceal those differences (e.g., a fallen tree, a bridge collapsed, or hidden adversaries). A UAV, with its greater speed, mobility and flight capability [1], can stay ahead of the ground vehicle and provide real-time imagery for route planning and mapping, enabling the ground vehicle to adapt its route as the situation demands.

In [2], we considered a precursor problem which defined a view corridor along and to either side of the upcoming route. The UAV provided complete coverage of the corridor, maintaining a position ahead of the moving ground vehicle. However, the speeds required from the UAV are very large. As a result, the ground vehicle must either be restricted to very slow speeds, or the UAV can cover only a narrow corridor around the planned route. In this paper we consider the problem in which the UAV covers only the parts of the corridor that cannot be observed directly from the ground vehicle due to occlusions.

As the ground vehicle moves through the terrain, any occluded parts of the corridor must be visited in advance by the UAV. See Figure 1. These hidden regions are approximated by finding areas that are never within the visibility polygon when viewed from locations along the ground vehicle's route [3], [4]. The planner must find a UAV path that covers as much of the hidden areas as possible within the resource and time constraints available: this is an Orienteering Problem with Time Windows (OPTW) [5]. Similar to the approaches of [6], [7], we cast a Mixed Integer Linear Program (MILP) to find a solution. Given that OPTW problems are proven to be NP-hard [8], we expect this formulation is likely to be NP-hard as well.

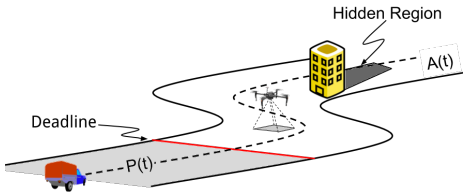

Fig. 1: The coverage corridor projected ahead of the ground vehicle with the observation area highlighted in light grey. The observation area moves forward with the ground vehicle.

In this paper we focus on the problem of planning coverage for a fixed ground vehicle route. We envision this solution as part of a receding horizon planner that replans the coverage path when the ground vehicle's route is updated due to newly acquired information on hidden regions. A complete solution requires a reliable data link between the UAV and the ground vehicle, to transmit updated waypoints to the UAV and send live sensor data back to the ground-vehicle; however, we leave this for future work.

II. Related Work

UAVs and ground vehicles frequently operate in collaboration to complete a task, as in exploration [9], persistent monitoring [10], and search and rescue [11]. Other applications combine a UAV's mobility with a ground vehicle's cargo capacity and take advantage of the strengths of both [12]. In some situations, a UAV may even serve as the ground vehicle's sensors [13].

In Orienteering Problems, an agent must collect some optimal subset of rewards given constraints on limited resources. The problem may be exploration with a UAV [14], optimal path planning for views [15], or team oriented planning [16].

Vehicle routing problems are well studied in the literature. Coverage planning such as [17], [18], [19] solves for complete coverage solutions and does not allow for partial coverage planning subject to limited resources (e.g., time, fuel, etc.). The Vehicle Routing Problem with Time Windows (VRPTW) studies planning for agents with constraints that limit the number of vertices to be visited before returning to the start, resulting in a route with multiple loops [7]. This

This research is partially supported by the Natural Sciences and Engineering Research Council of Canada (NSERC).

The authors are with the Department of Electrical and Computer Engineering, University of Waterloo, Waterloo ON, N2L 3G1 Canada (barry.gilhuly@uwaterloo.ca, stephen.smith@uwaterloo.ca)

type of problem focuses on optimal scheduling and minimal travel distances, not on maximizing a potential reward. In vehicle routing with time constraints [20], [21], an efficient heuristic solution is developed after discretization, reducing the problem to a directed acyclic graph. In our case, this approach isn't possible as the resulting graph would have cycles.

Contributions: There are three main contributions in this paper. First, we present the problem of providing UAV coverage path planning to a moving ground vehicle and solve it as an OPTW. A MILP is formulated to maximize the coverage area while minimizing UAV path length. Second, we prove this coverage problem, under certain conditions, can be exactly partitioned without loss of optimality, resulting in shorter and more predictable solution times. Third, we present a method of limited loss partitioning, for which the loss in solution quality is bounded.

III. Problem Statement

Let E be a planar environment in $\mathbb{R}^2$. The environment contains a set of m polygonal obstacles $O_1; \ldots; O_m$ that could be buildings, or other objects. Each obstacle is assumed to be tall enough that it blocks the ground vehicle's line of sight. The ground vehicle follows a collision free route $P(t)$ through E with velocity V_{gv} for $t \in [0; T]$ as shown in Figure 1. We define an area of the environment immediately ahead of the ground vehicle and centred on the route as the observation area, with width w and extending distance d in front of the ground vehicle. As the ground vehicle traverses P, the observation area moves forward as well.

The leading edge of the observation area is denoted as the deadline. As the ground vehicle traverses P, the deadline remains a distance d ahead. If we define $t = \frac{d}{V_{\text{gv}}}$ as the time required for the ground vehicle to traverse d, then the deadline is at $P(t_d)$ where $t_d = t + t$.

We define the region of E that is swept by the deadline from time 0 to a time $t \in [0; T]$ as the coverage corridor $A(t)$. The close time t_{close} for any point in $A(t)$ is defined as the first time the deadline sweeps over that point. An open time is also defined for each point, capturing the earliest time the point is usefully observed. The open time is relative to the velocity of the ground vehicle and the close time: $t_{\text{open}} = t_{\text{close}} \quad d/v_{\text{gv}}$.

Let $V(t)$ be the set of points in E visible by the ground vehicle when located at the point $P(t)$ on the path, as illustrated in Figure 2. A point $p \in A(T)$ is said to be *hidden* if it is has not been observed by the ground vehicle prior to the deadline passing it. We let R_{hidden} denote the *hidden region*, which is the set of all hidden points in the coverage corridor $A(T)$. More formally, consider a point $p \in A(T)$ and let t_d be the time when the deadline passes p. Then, p is hidden if $p \notin V(t)$ for all $t \in [0; t_d \quad t]$.

The UAV flies over E with velocity V_{uav}. We assume that the UAV flies at sufficient height that its environment is obstacle-free. Thus, it can travel over the obstacles $O_1; \ldots; O_m$. We assume that the UAV model has a steering function available that generates a path between any two points in E. The UAV has a downward focused camera providing coverage information to the ground vehicle. The functional size of the sensor footprint on the ground is $f \quad f$. Finally, we assume the velocity of the UAV is greater than that of the ground vehicle.

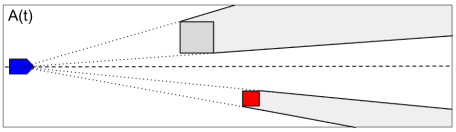

Fig. 2: Illustration of the visibility region for the ground vehicle, used to calculate the hidden regions.

With this background, our primary problem can be stated.

Main Problem III.1 (Selective Coverage Path)**.** Given a ground vehicle moving through the environment along the route P, plan a path for a UAV to maximize the area of the points within R_{hidden} that are covered within their time windows.

Solution Approach: Our approach starts by converting the continuous environment $A(T)$ into discrete cells and computing the hidden regions R_{hidden}. We then utilize sweep lines to cover each connected component of the hidden region. Next a graph with vertices representing sweep lines and edges representing transitions is constructed. We cast a MILP to solve for a path that maximizes the total coverage.

We develop a divide and conquer approach of partitioning the graph into subgraphs while provably preserving the overall optimal solution. We then describe a method of lossy partitioning that allows for even smaller subgraphs, while providing a guarantee on the maximum amount of lost coverage. The full UAV path is then constructed by concatenating the paths found in each subgraph in temporal order, starting with the earliest.

IV. Development of MILP Solution

We begin by describing the method for finding the hidden regions and placing the coverage sweep lines. Then, given the set of coverage lines, we derive a graph representation and subsequently solve for optimal coverage using a MILP.

A. Placing Coverage Lines

The environment E is first discretized into cells. The size of each cell is determined by the accuracy desired and the computational time available. Each cell uses t_{open} and t_{close} of its centre. Next the hidden regions, R_{hidden}, are calculated by sampling the views along the ground vehicle route P and converted into a cellular representation. A cell is considered to be within a hidden region if its centre is within the boundary of the region. Finally, the enclosing hulls are calculated for the cells in each hidden area. The result is a set of polygonal approximations of the hidden regions.

Parallel line segments, separated by f, are placed to provide complete coverage of each polygon found for the hidden regions. To minimize resource consumption [22], the segments are placed with an orientation that is perpendicular

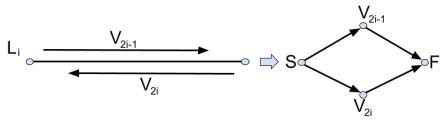

Fig. 3: Every coverage line L_i has two associated directional vertices in the graph representation, v_{2i-1} and v_{2i}. The UAV only visits one of these when moving from S to F.

to the minimum altitude line of each polygon [23], [18]. The resulting coverage lines, when followed by the UAV, completely cover the polygon. A brief greedy optimization is then applied to reduce the total coverage distance required.

The closing time for a sweep line is the latest time before which the UAV will be able to successfully cover that line. Closing times must take into account both the earliest closing time of all the cells that fall within the the UAV sensor footprint as the UAV transits the line, and the time required to transit the entire line. The closing time for the ith sweep line is defined as the minimum t_{close} of all the cells within $f=2$ of the line minus the UAV traversal time.

The opening time for a sweep line is the earliest t_{close} of all the cells within $f=2$ minus the time required for the ground vehicle to traverse the observation area. If the UAV arrives at a sweep line prior to this time, it must wait, adding idle time to the solution. Note that the length of any sweep line is limited such that $o_i < c_i$.

The reward R_i for sweep line i is the number of cells within $f=2$ of the line.

B. Graph Representation

The coverage problem is represented as a directional graph with time constraints, parameterized as $G = (V; E; s; t)$ with time windows $(o; c)$ and rewards R. There is a pair of vertices in V for each of the two directions that a line may be traversed (See Figure 3). There is also one vertex for each of the start and finish positions. The open and close times for each vertex are o and c respectively, and when referring to a vertex v we will write the times as v_{open} and v_{close}.

The edges E of G are not fully connected. Given two vertices a and b, then edge $\{a; b\}$ is added to the graph if vertex a and vertex b do not represent the same sweep line, and if $a_{\text{open}} + s_a + t_{a;b} \quad b_{\text{close}}$, where s_a is the service cost of a and $t_{a;b}$ is the transit cost for edge $\{a; b\}$. The service cost s_a is the time required to traverse the sweep line. Since visiting a vertex necessitates a physical transition in space, the travel costs between vertices are asymmetric. The last parameter, R, is the reward acquired for visiting a vertex.

NP-hardness: This formulation of the OPTW problem can be seen to be NP-hard as it contains the Euclidean TSP as a special case when the time windows for each vertex are $[0; 1)$ and the segment lengths are zero.

C. Formulation of the Mixed Integer Linear Program

To solve the Selective Coverage Path Problem on a given graph G, we formulate a MILP. Each pair of vertices representing a sweep line in G is identified sequentially $v_{2i-1}; v_{2i}$ for $i \in \{1; \ldots; \frac{|V|}{2}\}$. The start and finish vertices are v_0 and v_{N+1} where $N = |V|$. The variables of the MILP are defined as follows:

v_i - 1 if vertex i is in the solution, 0 otherwise, $i \in \{0; \ldots; N+1\}$.
x_{ij} - 1 if we travel from vertex i to vertex j, 0 otherwise.
u_i - service start time at vertex i.
R_i - reward collected from vertex i.
s_i - cost to visit vertex i (graph parameter s).
t_{ij} - travel cost for edge x_{ij} (graph parameter t).
$T_{\max}$ - total time budget, generally the time the ground vehicle arrives at the finish.
O_i, C_i - open and close times for vertex v_i.
M - a suitably large number ($2T_{\max}$ is used here).

The MILP has three objectives, which are solved in a hierarchical manner. Initially, we optimize for the collected reward (1). This produces a maximum coverage plan within the constraint imposed by the UAV's capabilities.

$$\max \sum_{i=1}^{N} R_i v_i \tag{1}$$

Subject to: (2)

$$u_{N+1} \quad T_{\max} \tag{3}$$

$$\sum_{i=1} x_{0i} = \sum_{i=1} x_{i(N+1)} = 1 \tag{4}$$

$$\sum_{i=0} x_{ik} = \sum_{i=1}^{N+1} x_{ki} = v_k; \forall k \in \{1; \ldots; N\} \tag{5}$$

$$v_{2k-1} + v_{2k} \quad 1 \quad \forall k \in \{1; \ldots; N/2\} \tag{6}$$

$$u_i + s_i + t_{ij} \quad u_j \quad M(1 \quad x_{ij}); \quad \forall i \in \{0; \ldots; N\}; j \in \{1; \ldots; N+1\} \tag{7}$$

$$O_i \quad u_i \quad C_i; \forall k \in \{1; \ldots; N\} \tag{8}$$

The amount of time for travel is limited and the plan must return to the finish before resources are exhausted (3). The solution must start at the initial vertex and end at the final vertex (4). For each vertex, there can be at most one connection in and one out (5). The solution has at most one vertex from each pair that corresponds to a coverage line (6). Subtours are eliminated and service times calculated by (7). Finally, visit times are confined to the specified time windows (8). The values of x_{ij} and v_i are binary variables, and the service times are positive real:

$$x_{ij}; v_i \in \{0; 1\}; \qquad u_i \in \mathbb{R}^+; \forall i \in \{1; \ldots; N\}.$$

The path found by the first optimization maximizes the reward, but there may exist a shorter path that collects the same reward. If time permits, the solution is reoptimized for length including the found maximum reward as a constraint. Finally, a third round of optimization minimizes the total service time u_{N+1}, removing any unnecessary idle time.

V. Partitioning to Improve Scalability

The exponential solution times of the MILP means that for many problem instances, an optimal solution is not found

before the solver exhausts its time budget. However, in this section we describe how we can exploit the structure of the time windows to partition G into subgraphs that can be solved independently.

A. Exact Partitioning

At any time $t \in [0, T]$, the vertices V in G may be divided into two sets

$$L_t = \{u \in V \mid u_{close} \leq t\}, \quad U_t = V \setminus L_t. \tag{9}$$

By convention, the start vertex is in L_t and the finish vertex is in U_t. The sets L_t and U_t can be said to partition V,

$$L_t \cap U_t = \emptyset, L_t \cup U_t = V, L_t \neq \emptyset, U_t \neq \emptyset.$$

Lemma V.1 (A Single Edge.) Given a time t, let

$$L_t = \{u \in V \mid u_{close} \leq t\}, \quad U_t = V \setminus L_t.$$

If for every $u \in L_t$, and $v \in U_t$, we have

$$v_{open} \geq u_{close} + s_u + t_{u,v}, \tag{10}$$

then in any optimal solution S there is exactly one edge between L_t and U_t, and it is directed from L_t to U_t.

Proof. We will prove by contradiction. Assume the route is cut at t and the inequalities in (10) hold. The vertices V of are partitioned into L_t and U_t. Consider any optimal solution S – there are two possible cases:

No vertices in U_t appear in S: If the optimal solution includes no vertices in U_t, then there are no edges from L_t to U_t. After visiting the last vertex $u \in L$, the UAV returns. However, by definition U_t is non-empty, and by (10), any vertex $v \in U_t$ can be visited from the latest closing time of any vertex in L_t. Therefore U_t is either empty or the proposed solution is not optimal, both of which are contradictions. Therefore, there must be at least one vertex from U_t in S.

There is at least one edge from U_t to L_t: For there to be more than one edge in S from L_t to U_t, there must be at least one edge from U_t to L_t. However, by (10), every vertex in L_t must have closed before any vertex in U_t opens. This case is not possible and there can be no edges from U_t to L_t. Therefore, there can be only one edge between L_t to U_t in S, and it is directed from L_t to U_t. □

From this Lemma we obtain a simple procedure for partitioning the graph into independent subgraphs. This method is detailed in the following.

Divide and Conquer Method:

1) Sort the vertices of G in order of v_{close}.
2) For each unique close time t, partition G into disjoint subgraphs L_t and U_t.
   a) If the conditions of Lemma V.1 are not satisfied, continue from Step 2 at the next t.
   b) Otherwise, add a dummy finish vertex to L_t with a zero cost from any vertex in L_t. Add a dummy start vertex to U_t with zero cost to any vertex in U_t.
   c) Store subgraph L_t as G_i, incrementing i.
   d) Continue from Step 2 with $G \leftarrow U_L$ until all possible partitions are tested. Store the remaining U_t subgraph as G_n.
3) Independently solve each stored subgraph, $G_1, G_2, \ldots G_n$. Construct the solution S by removing the dummy vertices of G_i, G_{i+1} and connecting the last vertex in the solution of G_i to the first vertex in the solution to G_{i+1} $\forall i = 1, \ldots, n-1$.

Proposition V.2 (Divide and Conquer Solution is Optimal) Any optimal solution of the Divide and Conquer method in Section V-A is an optimal solution to the Selective Coverage Path Problem III.1.

Proof. If the conditions of Lemma V.1 are satisfied, then the graph has the following properties that

The vertices of $u \in L_t$, $v \in U_t$ are strictly ordered:

$$u_{open} < u_{close} < v_{open} < v_{close}. \tag{11}$$

On any path from L_t to U_t, the UAV always arrives at $v \in U_t$ before or just as the time window for v opens. There are no edges from U_t to L_t.

Irrespective of the path selected in L_t, the UAV must always wait at the first vertex in U_t in the solution of S. Further, from (11) it can be seen that there are no edges (and therefore no paths) from U_t to L_t, no solution for U_t can affect the solution of L_t. Therefore, the two sections of the path are independent and may be solved separately. □

B. Limited Loss Partitioning

The conditions of Lemma V.1 are restrictive and frequently do not allow many partitions. This is particularly true when the environment contains a dense set of obstacles close to one another, and the resulting vertices in the graph have overlapping time windows. However, these are also regions in which partitioning would be most useful. Looking at the structure of the open and close times, there are locations in the graph where, if some vertices are "temporarily" removed, the conditions of (10) can be met. The temporarily removed vertices may not be included in the final result and so we define their associated rewards as the potential loss. Thus, we generate as many partitions as possible while bounding the sum of the potential losses as a function of the total path reward R_T. This observation leads to a limited loss method of applying Lemma V.1.

Limited Loss Divide and Conquer:

1) Sort the vertices of G in order of v_{close}.
2) For unique close times t_j, $j \in \{1, \ldots, n\}$ in G, find the number ℓ of vertices v with $t_{j-1} < v_{close} \leq t_j$, where $t_0 = 0$. Apply Lemma V.1 at t_j.
   a) If (10) is satisfied, then store $(\ell_j = \ell; c_j = 0)$ and continue from Step 2.
   b) Otherwise, temporarily remove the next vertex with close time $> t_j$ and mark it. Re-test (10). If the partition is not exact, repeat for the next

vertex until (10) is satisfied or no further vertices remain.

c) Evaluate the potential loss c_j as the sum of the rewards of the marked vertices at t_j. Store the vertex count and loss $(\ell_j = \ell, c_j = c)$ and continue from Step 2.

3) Find the subset of cuts using (ℓ_j, c_j) for $j = 1, 2, \ldots, n$ that minimizes the number of vertices in a subgraph, with a potential loss less than the prede ned fraction $\in [0, 1)$ of the total reward R_T in G.

4) Solve the subgraphs sequentially and in temporal order.

Remark (Temporal Order.) The subgraphs must now be solved in increasing temporal order as limited loss partitioning no longer results in independent subgraphs. The last vertex and visit time from the solution of the prior subgraph act as the starting vertex and time for the following subgraph, preserving the overall graph timing.

Remark (On Removed Vertices.) The marked vertices are not actually removed when the graph is partitioned in Step 3. Those vertices are included in the subgraph immediately following the boundary where their potential loss was identified.

C. A Dynamic Programming for Min-Max Partitioning

To optimize our limited loss partition method, we select the partition of G that is the most advantageous. Since the number of variables in the MILP grows by the square of the number of vertices in the graph, the best choice is the partition that minimizes the size of the largest subgraph while respecting the specified maximum loss. Selecting a subset of cuts from a list is similar to the dynamic programming 0-1 knapsack problem [24].

We start with a graph G and a list of n possible cuts (ℓ_i, c_i), each cut i having a length ℓ_i and a potential loss c_i. For any two cuts $i, j \in \{0, \ldots, n\}, i < j$, we define the length between cuts as the number of vertices from i to j, written $\ell_{i,j} = \ell_{i+1} + \cdots + \ell_j$. Given a loss budget $B = \epsilon R_T$, we seek the list of cuts $I = (i_1, i_2, \ldots, i_k)$, $i_1 < i_2 < \cdots < i_k$ where $\sum_{i \in I} c_i \leq B$, that minimizes the maximum length between cuts $\max\{\ell_{0,i_1}, \ell_{i_1,i_2}, \ldots, \ell_{i_k,n}\}$.

We define the subproblem

$$L(j, b) = \{\text{the min-max length of cuts up to cut } j, \text{ with budget } b\},$$

with the constraint that cut j must be included in the solution of the subproblem or the length is defined as ∞. The smallest subproblems which start the recursion are

$$L(0, b) = \begin{cases} 0 & \text{if } 0 \leq b \leq B, \\ +\infty & \text{if } b < 0. \end{cases} \tag{12}$$

All other subproblems are solved with the recursion

$$L(j, b) = \min_{i \in \{0, \ldots, j-1\}} \max\{L(i, b - c_i), \ell_{i,j}\},$$

and the final answer given by

$$\min_{j \in \{0, \ldots, n\}} \max\{L(j, b - c_j), \ell_{j,n}\}.$$

From the solution to the subproblems we extract the cuts that form the boundaries of the optimal partition of G.

Time Complexity: One complication in the application of dynamic programming to this problem is that all of the potential losses are real values, not integers. As a result, the size of the table required to look up repeated values is unbounded. In order to provide computational guarantees the potential losses are scaled to an integer value B' and rounded up such that $c_i' = \lceil \frac{c_i}{B} B' \rceil$. For a graph with $|V|$ vertices, scaling and rounding guarantees a solution with at most $|V| B'$ values and eliminates floating point errors. The value of B' is selected to be large enough that the results remain accurate, while limiting the cost of computation. After scaling, the final dynamic programming solution has time complexity of $O(|V|^2 B')$.

D. Proof of Limited Loss

Based on the discussion of Sections V-B and V-C we arrive at the following proposition. In this proposition we use $R(S)$ to denote the reward collected by a path S. We use R_T to denote the sum of all rewards in the graph:

$$R_T = \sum_{i=1}^{|V|} R_i.$$

Proposition V.3 (Limited Loss Partitioning.) Given a value $\in [0, 1)$, the Limited Loss Divide and Conquer method of Section V-B produces a solution S with reward

$$R(S) \geq R(S^*) - \epsilon R_T,$$

where $R(S^*)$ is the reward collected by the optimal solution.

Proof. Given a $\in [0, 1)$, the Limited Loss Divide and Conquer method produces a partition of G into k subgraphs $G_1, \ldots G_k$. Let $G_1', \ldots, G_k'$ be the subgraphs after removing the marked vertices (as described in 2b of the algorithm) from $G_1, \ldots G_k$. These marked vertices have a total reward of at most ϵR_T. The graphs $G_1', \ldots, G_k'$ satisfy Proposition V.2 and thus we can find a solution $S' = \{S_1', \ldots, S_k'\}$, obtained by concatenating the optimal solutions on each subgraph. We let $R(S_1'), \ldots, R(S_k')$ be the rewards collected by the S' in each of these subgraphs. The optimal solution S^* on G must contain some vertices in each G_i', call them S_i, such that $R(S_i') \geq R(S_i)$. That is, the optimal solution S^* may contain exactly the same vertices S_i' as or fewer vertices. Thus,

$$R(S') = \sum_{i=1}^{k} R(S_i') \geq \sum_{i=1}^{k} R(S_i). \tag{13}$$

The total reward for the optimal solution is

$$R(S^*) \leq \sum_{i=1}^{k} R(S_i) + \epsilon R_T \tag{14}$$

Therefore, after combining (13) and (14), the reward from partitioning G' is $R(S') \geq R(S^*) - \epsilon R_T$. Since each solution S_i' for the subgraph G_i' is a feasible solution in G_i (the graph G_i with the marked vertices included), it immediately

Fig. 4: The simulation environment, with the ground vehicle route running from top left to bottom right. Calculated hidden regions and the UAV's planned flight path are also shown.

follows that the optimal solution on $G_1, \ldots, G_k$ is at least that of the optimal solution obtained from $G'_1, \ldots, G'_k$. □

Remark (Conservativeness of Bound): As δ becomes larger, the bound in Proposition V.3 becomes very loose. To obtain the bound, we assume that S collects the entire R_T reward of the removed vertices. In practice this will not be possible. In addition, when implementing the Limited Loss Partition, because vertices are not actually removed, the solution on G can potentially collect rewards in addition to $R(S)$.

VI. Simulations and Results

We demonstrated selective coverage and the effectiveness of partitioning in a randomized software environment. At the start of each trial, the ground vehicle is given a map of the obstacles. A route is planned for the ground vehicle using an RRT* planner [25] and relayed to the UAV planner to calculate the obstacles and plan the UAV path. The planner must find a path that covers as much of the expected hidden regions as possible given the UAV's operational constraints. As shown in Figure 4, each environment is generated with 20 to 120 randomly placed obstacles. The start and finish positions are a constant 3000 m apart. The vehicle parameters are $V_{gv} = 1\,m/s$, $V_{uav} = 5\,m/s$, $w = 200m$, $d = 300m$, and $f = 25m$.

All simulations[1] are implemented in C++ using the following libraries and tools: Visibility polygons [26], Computational geometry functions from Boost [27], Minimum polygon heights [23], Concave polygons [28], QuickHull [29], and Polygon decomposition [30]. The Dubins curve library [31] provides our steering function for the UAV path, adding a radius of 1 m to all turns. We solved the MILP using Gurobi [32], allowing up to $t_{total} = 1000s$ on a Ubuntu 18.04 desktop PC with an Intel(R) i7-7700K CPU and 32GB RAM.

[1] A video demonstration of the simulation environment is available: https://youtu.be/mOyUuMXjomo

In our results, we present data from four different solution methods: Optimal, ExactDnC, DnC30 and DnC50. Optimal uses the unmodified graph G. ExactDnC allows partitions according to Lemma V.1. DnC30 and DnC50 allow lossy partitioning limited to 30 per cent and 50 per cent of the total coverage reward R_T, respectively. The results are grouped and plotted as a function of the number of vertices in G, which are directly proportional to the number of sweep lines in E. As the number of obstacles in the coverage corridor increases, so too does the number of sweep lines, indicating a greater difficulty in providing an optimal solution given a fixed time budget for the MILP solver.

Subdivision of Available Solution Time: The MILP solver is given a fixed time budget t_{total}, to be divided between the subgraphs. Since the number of variables in the MILP grows as the square of $|V|$ (see Section IV-C), we allot time proportional to the square of the number of subgraph vertices. Unused solution time is passed to the next subgraph.

A. Exact and Limited Loss Partitioning

Figure 5a compares the time required to find a solution to the MILP using the four different solution methods as a function of the number of vertices in the full graph G. After five hundred simulations, the results are collected and plotted by vertex counts. The plots show the mean value and 95% confidence interval for each vertex value. Initially, all methods terminate prior to their time budget. However, as the number of vertices increases, the time required to solve either of the Optimal or ExactDnC cases quickly reaches the maximum limit. There is large variability in solutions times, as Gurobi solves some instances very quickly, while others of similar size use the entire time budget. This commonly occurs when solving NP-hard problems. The lossy methods, DnC30 and DnC50, are able to solve much larger problems within the allotted time. Figure 5b plots the number of covered cells in simulation, when following the UAV path generated by the planner. The simulations are consistently close to the expected MILP reward.

B. Path Lengths

In this experiment we ran 1000 trials, increasing V_{uav} to 20 m/s and limiting the MILP to $t_{total} = 120$ s. The results can be seen in Figure 5c. Under these conditions, as the scale of the problem increases, the Optimal and ExactDnC solutions are clearly not capable of minimizing the length of the path in the available time.

VII. Conclusions and Future Work

We formulated a MILP that solves the Selective Coverage Path Problem. We then proved that the coverage problem can be exactly partitioned while retaining the optimal solution under certain conditions. We also developed a limited loss partitioning method which found an optimal partition within a bounded loss in coverage reward. The lossy method allowed the coverage problem to be subdivided in cases where the exact partition was not possible. Lossy partitioning may also provide better solutions for scenarios where the solver