Regular Article

Large Scale Aerial Multi-Robot Coverage Path Planning

Kunal Shah[1], Annie E. Schmidt[2], Grant Ballard[2] and Mac Schwager[3]
[1]Department of Mechanical Engineering, Stanford University
[2]Point Blue Conservation Science
[3]Department of Aeronautics and Astronautics, Stanford University

Abstract: Autonomous survey and aerial photogrammetry applications require solving a path planning problem that ensures sensor coverage over a specified area. In this work, we provide a multi-robot path planning method that can obtain this coverage over an arbitrary area of interest. We extend our previous method, path optimization for population counting with overhead robotic networks (POPCORN), by a divide-and-conquer scheme, split and link tiles (SALT), which drastically decreases the time needed for route planning. These POPCORN instances can be computed in parallel and combined with SALT in a scalable manner to produce coverage paths over very large areas of interest. To demonstrate this algorithm's capabilities, we implemented our planning algorithm with a team of drones to conduct multiple photographic aerial wildlife surveys of the Cape Crozier Adélie penguin rookery on Ross Island, Antarctica, one of the largest Adélie penguin colonies in the world. The colony, which contains over 300,000 nesting pairs and spans over 2 km, was surveyed in about 3 hours. In contrast, previous human-piloted single-drone surveys of the same colony required over 2 days to complete. We also have deployed our survey system at several islets at Mono Lake, California, to survey a California gull colony as well as at a 2000-acre ranch in Marin, California. We provide this survey path planning tool as an open-source software package named `wadl`.[1]

Keywords: aerial robotics, environmental monitoring, motion planning, navigation

1. Introduction

In this work, we present a new set of methods to solve large scale path planning problems for aerial surveys with multiple autonomous uncrewed aerial vehicles (UAVs). Field missions that involve the physical coverage of a target area, such as photogrammetry or search-and-rescue, fundamentally require solving the coverage path planning (CPP) problem, which finds paths over the area such that the agents traverse the entire area while providing sensor coverage at a specified resolution. While there are many uses for CPP (Cabreira et al., 2019; De Carvalho et al., 1997; Englot and Hover,

[1] https://github.com/k2shah/wadl

Received: 19 October 2021; revised: 10 September 2022; accepted: 9 October 2022; published: 7 November 2022.

Correspondence: Kunal Shah, Department of Mechanical Engineering, Stanford University, Email: k2shah@stanford.edu

DOI: https://doi.org/10.55417/fr.2022064

2012; Oksanen and Visala, 2009), the focus of this work is developing a CPP method for large scale aerial surveys in remote areas where direct access may not be possible on foot. Furthermore, due to weather variability, lighting variability, and dynamic conditions on the ground, the survey should be conducted as quickly as possible. In addition to their massive commercial applications (Otto et al., 2018), UAVs have become a popular tool allowing researchers in various disciplines to gather data over field sites. As survey sites of interest become larger and more remote, multi-agent path planning solutions present an effective way to scale survey tasks without long multi-day operations. Our goal is to develop a cohesive system to fully utilize the capabilities of a multi-UAV system for coverage tasks over large areas and in extreme environments.[2]

The method presented here, split and link tiles (SALT), improves our previous method, path optimization for population counting with overhead robotic networks (POPCORN) (Shah et al., 2020) by addressing a key scalability issue. After a certain size POPCORN is unable to find routes within a reasonable amount of time due to the exponential complexity. This proved to be a large issue during field operations where minor changes in the survey boundary caused large wait times. To combat this, we use the POPCORN method to solve small subproblems of the total coverage path planning problem and use SALT to combine these paths in a scalable manner amenable to single or multi-agent planning instances. POPCORN is a Boolean satisfiability (SAT) relaxation of the traveling salesman problem (TSP) in which we allow node revisitation. SALT first partitions a graph that represents the desired survey area into smaller graphs which we refer to as "tiles." We show that the paths through each tile using POPCORN can induce another graph, we which refer to as a pathgraph. We detail novel graph partitioning strategies that can be used on the induced graph to effectively stitch together the path through each tile to find a set of flight paths for the entire team of UAVs to complete the survey. The key feature of SALT is that it takes a large, exponentially complex problem instance and converts it into a series of smaller, complex but solvable instances, and combines these smaller solutions to find the final paths.

It is not always possible for field crews to remain at remote deployment sites for long periods of time. For instance, crews may be transported to a site via helicopter and have just a few hours to complete survey tasks. Some survey planning efforts have proven difficult with POPCORN alone since solution times could take upwards of 50 hours, which can be prohibitively costly in deployment scenarios where minor modification, such as geofence adjustments, could render entire deployments inoperable. However, POPCORN+SALT has dramatically improved the solution time of survey planning with little impact on overall performance; in some cases solution times have been reduced from 50 hours to 5 minutes. As the size of the desired survey area grows, the complexity for a single POPCORN instance also grows exponentially. SALT addresses this problem by keeping the problem complexity to a manageable level at the cost of suboptimal paths, which can be seen in Section 4. Despite this problem, POPCORN+SALT is still able to produce actionable flight paths that take less overall flight time when compared against classic sweep style paths or more modern route planning methods like Google's OR-Tools (Perron and Furnon, 2019). Furthermore, since the underlying planning problem is NP-hard it is unclear if finding the optimal path is even possible within a reasonable amount of time. Using novel graph partitioning methods, our method can find paths for large areas fast enough to be used for on-the-fly route planning scenarios.

In this paper, we describe two graph partitioning algorithms used to link together subpaths: a breadth-first-search (BFS) method, which is fast and guaranteed to find a solution, and a mixed-integer linear program (MILP) that is designed to minimize the total number of routes. The BFS method, while fast, is a greedy method that may not always find the smallest number of paths. While the MILP method can find the smallest number of routes, it can be slow and suffers from the standard drawbacks of any MILP/IP solution method. The addition of SALT has allowed various field teams to efficiently survey large areas with multiple drones in a matter of hours. These methods

[2] A video overview of the paper can be found here: https://youtube.com/playlist?list=PLl_DVpWDF4k9WmnMaJLLekcSIjXAublnM.

are implemented in our open-source package `wadl`, and can enable finding routes for areas upwards of 10 km^2 in a matter of minutes.

Readily available commercial coverage planning methods, software packages, and algorithms focus on solving the planning problem over the survey area with a single UAV (Cabreira et al., 2019; Choset, 2001; Galceran and Carreras, 2013; Pix4d Inc, n.d.; SPH Engineering, n.d.). These methods do not consider the time it takes to travel to and from the UAV launch point. Without this transit consideration, popular methods like the classic "sweep" method result in large amounts of backtracking over the survey area. Furthermore, these tools assume only one UAV is needed, and thus only a single path is generated. Naïvely dividing this single path can lead to large amounts of wasted flight time since each path over the survey region would potentially start at an arbitrary point. We develop a unified tool designed for one or more UAVs to complete an aerial survey over any size area while accounting for the practical constraints present during field operations. Our method focuses on finding the shortest overall time it takes to complete a survey by reducing the backtracking over the survey area as well as minimizing the transit time to and from the launch site. We also provide a method designed to reduce the total number of individual flights, as each additional flight may contain indirect costs such as needing extra batteries, personnel, or other logistical requirements.

Compared to one state-of-the-art scheduling package, Google OR-Tools (Perron and Furnon, 2019), we find that POPCORN+SALT finds routes that take 15% less total flight time given the same computation time. We also demonstrate our method with a series of case studies that illustrate its effectiveness with a team of UAVs to survey Adélie penguin (*Pygoscelis adeliae*) rookeries on Ross Island, Antarctica (77.455°S, 167.215°E) and California gulls (*Larus californicus*) at Mono Lake, California, USA (38.018°N, 119.044°W) as well as a survey of a ranch in Marin County, California, USA.

Related Works

Coverage path planning has been an active area of research for many decades (Choset and Pignon, 1998) and has seen many recent applications (Moon and Shim, 2009; Otto et al., 2018) given the affordability and availability of commercial UAVs. Commercial software packages are designed for single agent coverage planning, and use a sweep or "lawnmower" flight pattern, despite there being more optimal planning methods (Cabreira et al., 2019; Choset, 2001; Galceran and Carreras, 2013). While sweep paths are the de facto standard in many applications, in situations where the UAVs' launch points are physically far from the survey area, there can large sections retraversed when the UAVs travel from the launch site to the survey area, and back. This backtracking behavior can lead to longer overall survey times and increases collision probability if multiple UAVs are in flight simultaneously (Shah et al., 2020). Other methods for CPP use a simple geometric pattern, like a sweep, spiral, or space-filling (Hilbert) curve or, for larger areas, use geometric or "cellular" decomposition (Acar et al., 2002; Huang, 2001; Kong et al., 2006; Oksanen and Visala, 2009) to partition the space into cells. For these decomposition methods, sweep, or similar, paths are planned in each cell that are later recombined with geometric or graph search methods (Bähnemann et al., 2021). However, while these single agent techniques are fast, they perform poorly in multi-agent planning tasks as they directly distribute a single path to multiple agents without considering ingress locations. The review (Almadhoun et al., 2019) gives an overview of the current state of multi-agent coverage path planning. Single agent coverage methods using minimal spanning trees (MSTs) (Agmon et al., 2006; Gabriely and Rimon, 2001) have been extended to work well in the multi-agent case (Kapoutsis et al., 2017) by using a Voronoi partition to divide the coverage tasks among multiple agents. In other works (Zheng et al., 2005), the MST can be partitioned directly, but the paths still can contain significant backtracking. However, while these methods succeed in finding a multi-agent solution, they assume that the agents can start anywhere in the area of interest, which is not always possible in some field applications. In other cases, genetic algorithms have been used with some success (Kapanoglu et al., 2012) but it is hard to include certain route limit constraints (such as battery life) into these methods.

Instead of employing a geometric pattern, we encode the path planning problem as a discrete planning problem over a graph. Nodes in this graph represent points in physical space and every point must be visited to achieve complete coverage. Solving for a cyclic path through the coverage graph in the single-robot case can be converted to the classic TSP (Held and Karp, 1970, 1971; Gutin and Yeo, 2007). To relax the problem we allow for node revisitation in our formulation, since not all graphs have a TSP solution, e.g., a line graph. The TSP has a multi-agent generalization, known as the vehicle routing problem (VRP) (Bektas, 2006; Toth and Vigo, 2002), which is designed for routing vehicles through cities. Methods available for solving the TSP and VRP often resort to heuristic methods for large problems (Perron and Furnon, 2019) due to their exponential growth with problem size and focus on finding optimal solutions that are constrained to visit every vertex only once. Some existing multi-agent survey planning works cast the problem as a VRP and solve it using these existing tools (Jing et al., 2020; Perron and Furnon, 2019).

We choose to model and solve the multi-agent CPP problem using SAT. The added flexibility afforded by the SAT framework allows us to model high-level coverage constraints (i.e., to visit all nodes in the coverage graph) alongside lower-level dynamics constraints (i.e., only move between adjacent nodes). SAT methods have been used for a variety of task and motion planning problems (Hung et al., 2014; Imeson and Smith, 2019; Scala et al., 2016; Shoukry et al., 2016). Similar to Surynek (2014, 2015) this work uses one-hot style encoding to translate the multi-robot path planning problem into a SAT. These works formulate a SAT problem to find paths for robots to move to fixed designated goal positions under collision avoidance constraints and, like our method, use an off-the-shelf SAT solver (De Moura and Bjørner, 2008).

We also employ a graph partitioning method that helps to address the scalability issues that arise with large discrete path planning methods. Graph partitioning problems seek to divide a graph into two (Kernighan and Lin, 1970) or more (Andreev and Racke, 2006) sets of vertices whose union is the original vertex set. We develop a breadth-first search graph partitioning method that produces a set of trees to cover a parent graph which all share the same root node and are under a maximum total edge weight cost. We also present a MILP method to find the minimal number of partitions inspired by supply-and-demand network problems (Jovanovic and Voß, 2016). While a MILP is still a combinatorial optimization problem, it is much smaller than the original problem and still performs well for reasonable sizes of survey areas.

2. Coverage Path Planning Formulation

We formulate the coverage planning problem in an arbitrary polygon for a multi-robot system as a discrete path planning problem on a graph. To encode the coverage constraint, we overlay a lattice over the survey area to generate an embedded graph $G(V, E)$ we refer to as a coverage graph. Each node $v \in V$ in this graph represents a point in physical space and can be mapped $L : V \to \mathbb{R}^2$ to a unique Cartesian coordinate in the survey area. An edge $e = (v_i, v_j) \in E$ implies that an agent can move from v_i to v_j. For simplicity, in this work we assume all coverage graphs are grid graphs and are connected, but not necessarily fully connected, graphs. The main goal of the planning algorithm is to assign the nodes v_i to K sets $V^1, \ldots, V^K$ to form K paths $p_1, \ldots, p_K$ such that all the nodes are visited $\bigcup_{k=1}^{K} V^k = V$ at least once. We impose a series of constraints on the paths to account for the practical environmental limitations present in field operations.

We assume that the coverage requirement is known, meaning that the grid spacing is derived from some other requirement. In the case of aerial photogrammetry, the agents take images from downward-facing cameras which are later stitched together with photogrammetry software (Agisoft, 2019; Microsoft, n.d.) to generate one large georeferenced image mosaic. The overlap requirements are specified by the photogrammetry software; for example, Agisoft's Metashape (Agisoft, 2019) requires at least 60% overlap. Determining the correct spacing for a given application is a function of the required image overlap between adjacent images, camera properties, and the UAV's speed and altitude above ground level (AGL) (Di Franco and Buttazzo, 2015; Nam et al., 2016). Typically a ground sample distance is also specified as a value in centimeters per pixel.

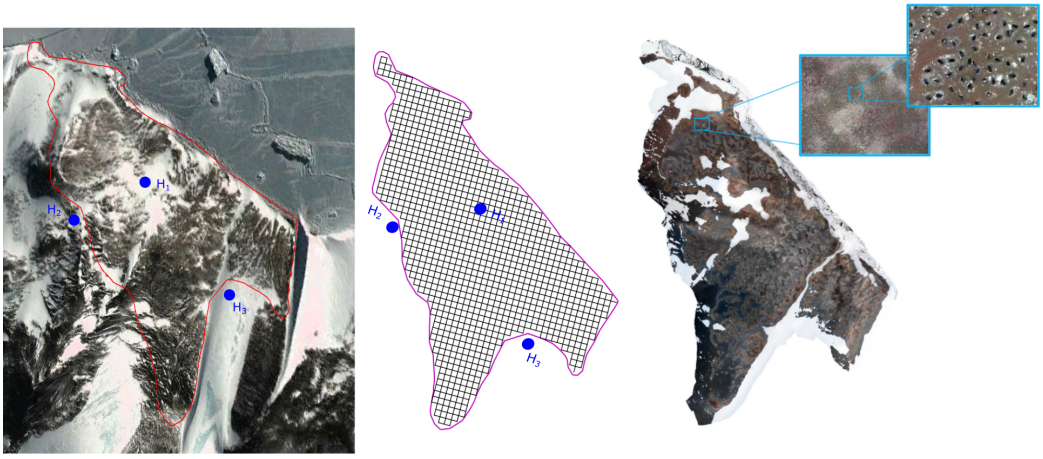

Figure 1. A satellite image of the West Cape Crozier Adélie penguin colony with the survey geofence and three home (H_i) locations (left). The coverage grid is found by overlaying a grid and intersecting it with the geofence (center). The images captured during the survey are then stitched together and used for population analysis; an individual image as well as expanded images of nesting Adélie penguins can also be seen (right).

Direct access to a field site may not always be possible. For regions that have densely populated wildlife sensitive to UAV take-off and landing or for areas that are not easily traversable (thin ice sheets, lake islands, steep terrain) it may only be possible to access these sites from remote "home" locations $\mathbf{H}$ outside the survey perimeter. In general, there can be multiple home points which represent distinct locations where it would be favorable to launch from (e.g., near a road, high location, near a power source, etc.). We directly incorporate these home constraints into our planning formulation because locations may be physically far from the survey location, requiring the UAVs to transit over large distances to access the survey site. While there is no requirement that the home locations $H_i \in \mathbf{H} \subset \mathbb{R}^2$ be inside or outside the survey perimeter, we require that a path p_k begins and ends at the same H_i. Each path p_k can be described by a home point $H^k \in \mathbf{H}$ and a set of nodes V^k:

$$p_k = (H^k, V^k = \{v_1^k, \dots v_n^k, \}) \text{ where } v_i^k \in V \text{ and } (v_i^k, v_{i+1}^k) \in E, \tag{1}$$

in the coverage graph G. Figure 1 shows a survey region with the associated grid and potential home sites as well as the final mosaic.

In order to decrease flight times over the survey area, we require that the paths over the survey be cyclic or "closed," meaning $v_1^k = v_n^k$. This can save considerable time in survey operations (Kapoutsis et al., 2017; Shah et al., 2020) as traditional survey methods (Galceran and Carreras, 2013) and commercial methods (DroneDeploy, 2021; Pix4d Inc, n.d.; SPH Engineering, n.d.) tend to significantly backtrack over the survey regions using "open" paths. This can cause significant wasted battery life, especially when the UAVs have a speed limit within protected areas. Figure 2 shows a side-by-side comparison of an open and closed path routed over the same area and shows the large extra section of path due to backtracking. The main goal of this paper is to formalize a method that reduces the overall length of a survey path while still meeting the coverage goal. This is done by finding cyclical paths through the coverage in order to reduce the amount of backtracked and wasted flight time.

Each path p_k has some physical path length s_k over the survey area,

$$s_k = \sum_{i=1}^{n} \|L(v_{i+1}) - L(v_i)\|, \tag{2}$$

which is the total length flown on the lattice. The UAVs also start and end at the home point H^k which is some distance h_k from the survey. We can find the shortest distance between any home $\mathbf{H}$

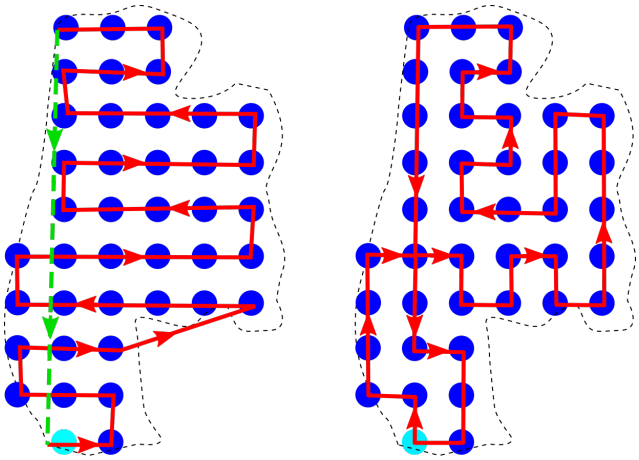

Figure 2. An example planning scenario showcasing the difference between an open and closed path. For the same planning instance we see a set of paths in red over a survey area with coverage grid in blue. An open path with green backtracking section (left) and closed path are shown over the survey area (right). Although each path starts from the same cyan point, we see that the closed path satisfies the coverage requirement with a shorter path when compared to the open path.

and any survey point in the path. Since the paths form a cycle, we can shift the index of the survey path and "attach" the home point:

$$h_k = \underset{H^k \in \mathbf{H}}{\operatorname{argmin}} \min_{v \in V^K} \|H^k - L(v)\|. \tag{3}$$

Each flight begins when the UAV takes off from the home location then the UAV flies the distance h_k to the survey site and completes the survey by taking images. After the data are collected the agent returns to the home location and lands, signaling either a battery exchange for another flight or the end of the survey. We assume the UAV flies between the home and survey locations at a known speed v_h and flies the survey at v_s. We also assume that the maximum usable flight time (T_{max}) is known. For simplicity we assume that these parameters are the same over the entire fleet of UAVs. Thus each path has the constraint that it must be completed within the time limit,

$$t_k = 2\frac{h_k}{v_h} + \frac{s_k}{v_s} \leq T_{max}, \tag{4}$$

where t_k is the total time it takes to complete path p_k. The term $2\frac{h_k}{v_h}$ is the time it takes for the UAV to fly to and from the survey from the home, covering the distance of h_k flying at speed v_h. The term $\frac{s_k}{v_s}$ is the time the UAV takes to fly the survey path distance of s_k at speed v_s. The sum of these terms must be less than the flight time allowed by the UAV's battery, $t_k \leq T_{max}$. For survey locations that are physically far from the home location, inefficient routing can result in long survey times, which are further made problematic in areas with short weather windows of acceptable flight. We seek to find a set of assignments to paths such that the total flight time is minimized,

$$\begin{aligned} &\min \sum_{k=1}^{K} t_k \\ \text{s.t.} \quad &t_k < T_{max} \\ &\bigcup_{k=1}^{K} V^k = V \end{aligned} \tag{5}$$

subject to the flight limits and coverage constraints. This problem is equivalent to the capacitated vehicle routing problem and is known to be NP-hard (Toth and Vigo, 2002). In some survey scenarios,

there may be other time costs associated with each flight (e.g., battery swap, battery charge, etc.). It is also beneficial to reduce not only the total time for the survey but also the total number of flights, K:

$$\begin{aligned} & \min K \\ \text{s.t.} \quad & t_k < T_{max} \\ & \bigcup_{k=1}^{K} V^k = V. \end{aligned} \tag{6}$$

Survey tasks over large regions motivate the need for scalable solution methods that optimize both total flight time as well as the total number of flights.

3. Path Planning Algorithm

Previously discussed methods for both single-agent and multi-agent planning do not always take into account range or battery constraints. Some commercial methods are designed so that a UAV returns home partway through a route to exchange batteries, and then goes back to where it left off to continue the survey. We design our planner with these range constraints in mind, and encode the geographic constraints directly into the planning process. Our goal is to produce a tool that can generate coverage paths for any sized area and for any number of agents. We output a set of paths which can either be done sequentially by one agent or divided among a set of agents and executed in parallel.

Given that our planner is grid based, as the size of the area increases, the number of grid points grows proportionally, leading to an exponential growth in possible paths. Like many methods that suffer from this type of growth, we employ a divide-and-conquer method that divides the coverage graph into a set of smaller graphs, akin to the cellular decomposition (Acar et al., 2002; Choset et al., 2000; Huang, 2001) methods used by many coverage planners. We use a SAT method to find a cyclical path through each smaller grid. Inspired by the cycle basis property of graphs (Diestel, 2004), we combine adjacent grids together in a way that preserves the cyclical path property of the whole path. This cycle-preserving property is responsible for the majority of the backtracking reduction when compared against traditional methods. We present two heuristic methods to solve the minimum time problem (5) that both share the same initial divide step, with one specifically tailored to solving (6).

Our method first overlays a grid over the desired geofence. This grid is then *split* into smaller grid sections referred to as "tiles." Using a SAT-based method we then find a closed (cyclic) path through each tile. After these paths are found, we *link* them together while respecting the UAVs' flight time constraints to find the final set of feasible routes that the UAVs will fly. A visualization of the route planning process can be found at the documentation[3] page of our codebase.[4]

Split

We employ a simple rectangular decomposition to divide the coverage graph G into smaller tiles. While other types of decomposition methods have been used (Azpúrua et al., 2018), each tile T_i has a grid graph with nodes $V_i \subset V$ (a subgraph of the coverage graph) associated with it. After these tiles are found we can solve for a path through each tile's grid graph as shown in Figure 3.

The split step first overlays a larger grid over the rectangular extends of the geofence. At each node in the coverage graph the UAV is commanded to take an image. We can find the size of the ground imaged as a function of the camera properties and the UAV's altitude (Di Franco and

[3] https://wadl.readthedocs.io/en/master/?badge=master
[4] https://github.com/k2shah/wadl

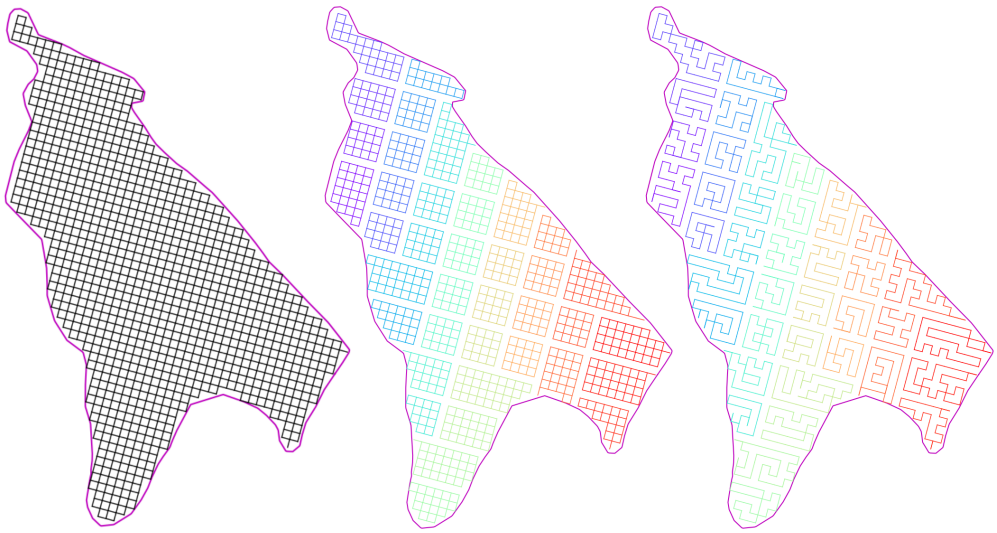

Figure 3. Step-by-step process of finding a set of coverage paths over an area from left to right. First, a coverage graph inside the desired geofence is generated (left). Then, this coverage graph is split into smaller tiles (center) and finally a path is found for each tile's grid (right).

Buttazzo, 2015; Nam et al., 2016). Since the survey altitude is known at planning time, the size of each survey image is also known in meters. Taking the smaller dimension of the image (since most camera senors are rectangular), we can expand the geofence by a fixed amount before we generate the paths. This way we can ensure that the union of all the images covers the desired geofence.

Minimum, nominal, and maximum tile sizes can be set by the user before the plans are found. The split method first divides the coverage graph into a number of tiles based on the nominal size and then will merge smaller tiles with sizes less than the minimum into adjacent tiles as long as it does not surpass the maximum size. If the tile size is too large, then the SAT solutions can take a prohibitively large time. Tile sizes should also be based on the survey flight speed and grid spacing, since each UAV path will fly through an integer number of tiles. For our flight planning tasks we aim to have each UAV cover between four and six tiles, depending on how far the launch points are from the survey location. We find that tile sizes between 30 and 50 work well for modern laptops used in the field.

The paths are designed to be a closed circuit that passes through each node as least once. This is a relaxed version of the TSP. We use a one-hot encoding scheme similar to Surynek (2014, 2015) to transform the path planning subproblem into a SAT problem as found in Shah et al. (2020) and summarized here. Let

$$X_{v,t} \in \mathcal{B} = \{0, 1\} \tag{7}$$

where $v \in V_i = \{1, \ldots, N_{vertex}\}$, and $t \in \{1, \ldots, N_{path}\}$ for a coverage graph with N_{vertex} nodes. If $X_{v,t} = 1$, it implies that the agent is at vertex $v \in V$ at step t. We can encode the designed behaviors by forming various logical statements in either prepositional or predicate logic. The solver attempts to find an assignment (either 0 (False) or 1 (True)) for each Boolean variable, $X_{v,t}$, such that the logical statement evaluates to True. These logical statements can be thought of as constraints similar to how an optimization problem is formulated.

To constrain the agents to only be at one node v for any given step t, we first require

$$X_{v,t} = 1 \; \forall \; t \; \exists! \; v, \tag{8}$$

where $\exists!$ requires there exists one and only one v. For a given t, this constraint forces there to be only one v such that $X_{v,t} = 1$, meaning that at any step t the agent exists at only one vertex v.

Next, we encode the constraint that agents can only move between connected vertices:

$$\neg X_{v,t} \bigvee_{y \sim v} X_{y,t+1}, \tag{9}$$

where $y \sim v$ implies that there is an edge $(v, y) \in E$; we allow self-connection, e.g., $(v, v) \in E$. If $X_{v,t} = 1$ then negating it and applying an "OR" with all vertices y connected to v ($y \sim v$) will force at least one $(y, t + 1)$ to be 1. This coupled with the previous constraint will force only one such $X_{y,t+1} = 1$; thus agents can only move between connected nodes. Finally, we encode the coverage constraint that every vertex v is occupied at least once:

$$\exists\ t \mid X_{v,t} = 1\ \forall\ v. \tag{10}$$

If we strictly enforced there to be only one t, then this would be equivalent to the TSP. This constraint encodes that for every v in the graph there is some t or step where the node v is visited, thus satisfying the converge requirement.

Since the path is a closed loop, we do not need to explicitly choose a particular start and end point. As such, let v^0 be an arbitrary vertex in the tile's grid graph,

$$X_{v^0,1} = X_{v^0,N_{path}} = 1, \tag{11}$$

which will serve as the start and end point of the path. This constraint implies that we have some node v^0 in the tile that will serve as the start and end of the survey path. Normally, this will be the physically closest vertex to the take-off location of the drone.

Binary variables that represent unreachable states can be removed from the problem to reduce the solution time. Furthermore, the SAT instances do not optimize an objective function directly and instead only find an any-time feasible solution. To find the shortest path, we can initially set $N_{path} = N_{vertex}$ since we need a minimum of N_{vertex} steps for a graph with N_{vertex} nodes. If this is unsatisfiable, we successively increase N_{path} according to some search schedule until the problem is feasible. This process can also be done in reverse by setting N_{path} to be initially much larger than N_{vertex} and then successively decreasing it until the problem is infeasible. A binary search can also be used. All these methods work well, but we find that the successive increments find smaller paths overall without much more computational overhead. Lastly, noticing that the same graph structure is found in many interior tiles, we can cache the solution of one and reuse it, saving some computation. Figure 3 shows a set of tiles with solved paths, with some having the same path modulo a rotation.

Although determining the number of steps needed for each tile directly is not possible, we can bound the number of steps needed to narrow the search space. The number of nodes in a tile can be used as a lower bound since, at minimum, the path must visit all nodes. In some cases this lower bound is exactly the number of steps needed for the path; this is equivalent to finding a Hamiltonian cycle of a graph. Unsurprisingly determining if a graph has a Hamiltonian cycle is an NP-complete problem. We can also upper bound the number of steps needed by using a minimum spanning tree (Held and Karp, 1970, 1971). In the worst case, the number of steps needed to cover a graph is equivalent to traversing the MST in a depth-first manner which takes exactly $2 * |E|$, where $|E|$ is the number of edges in the MST. This upper bound is realized in the case of a linear graph, where all the nodes can be considered to be in a "line" such that all nodes have degree 2 except for two nodes at the "ends" which have degree 1. From Figure 4 we can see that the shortest cyclical

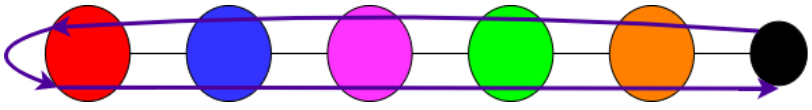

Figure 4. A linear graph. This figure a worst-case tile in which the number of steps needed to produce a cycle path through the graph is twice the number of edges in the graph. The shortest cycle (purple) starts at the black node and moves right through each node to the red node and then finally returns to the black node.

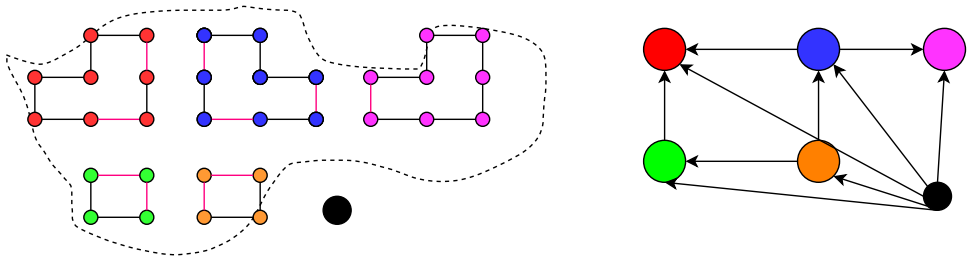

Figure 5. Unlinked tiles and the path graph. This figure shows the intermediate state before linking each of the paths found for each tile. A set of unlinked tiles with paths with home point in black (left). Edges directed away from the black home node are the home edges and the edges between the colored nodes are the base edges (right).

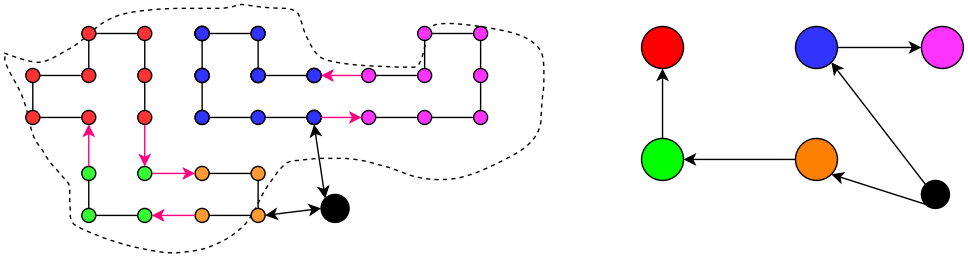

Figure 6. Linked tiles and the path graph. This figure shows the final step of finding the linked paths using the path graph. A set of linked tiles with paths transiting from the home point in black (left). The corresponding set of paths in the path graph. Two paths are found overall, one by linking the orange, green, and red tiles and another by linking the blue and magenta tiles (right).

path through this graph starts at one nodes, moves to one end, and backtracks over all the nodes. Without extensive analysis of the coverage graph, it is difficult to determine a tight upper bound as some survey areas could be a large open field such as our ranch survey (Figure 23) or could have tight constricting corridors like the northwest section of our Cape Croizer survey (Figure 1).

Link

After a path through each tile is found the next step is to "link" together adjacent tiles to form complete paths. We observe that two adjacent tiles can be joined together to form a larger cycle, similar to how cycles can be combined together in a graph via the algebra induced by its cycle basis (Diestel, 2004). Note that if two adjacent tiles share a pair of parallel edges, then those tiles can be joined such that the cyclic path through both tiles is preserved, which can be seen in Figures 5 and 6. Using this "path-adjacency" property we can combine tiles to form complete paths that are still cyclic and reduce backtracking. We now formalize a method to generate a set of K paths such that each tile is visited exactly once, and thus the whole graph is covered.

Each tile $T_i \in \mathbf{T}$ has an associated travel cost z_i and base cost b_i each measured in time (seconds). Time is used because the transiting can sometimes be flown at a higher speed than the actual survey (base) cost, so measuring time makes both costs comparable. The base cost b_i is found by simply taking the path length of the path on tile T_i and dividing it by the survey speed v_s; e.g., if a tile has a path length of 400 m and the survey UAV flies at 5 m/s then $b_i = 80$. Transit costs, z_i, are found by taking the shortest distance between any node in T_i and any home point H_i and dividing by the transfer speed v_h.

After the tiles are found, we generate a new weighted "meta graph" $Q(\mathbf{T} \bigcup Z, D, W)$; we refer to this graph as the path graph (Figures 5 and 6). The nodes in Q are the set of tiles $\mathbf{T}$ and Z which is an abstract node made to represent the home location. For each tile T_i we add a directed edge

$d_{Z,T_i} = (Z, T_i)$ from Z to T_i with cost $w_{Z,T_i} = 2z_i + b_i$. Recall that z_i is the time it takes to fly from a home location to the closest point in tile i and b_i is the time it takes to survey tile i. We refer to these edges as the "home" edges. In general, Z represents a set of home points, but the cost w_{Z,T_i} for a particular T_i represents the amount of time it would take to fly to and from the closest home point to T_i and survey all the points in T_i. Next we add a directed edge between two path-adjacent tiles $d_{T_i,T_j} = (T_i, T_J)$ with cost $w_{T_i,T_J} = b_j$ if $z_i < z_j$. We refer to these edges as the "base" edges. This edge allows for a path between two tiles directed away from the tile that has the lesser transit cost. By this construction the path graph is a directed acyclic graph (DAG) rooted at Z.

A single path P_k can be represented as a set of tiles S_k and has cost equal to the total time t_k required for that flight as in (4). This time cost can be written as $t_k = \sum_{i \in S_k} b_i + 2 \min_{S_k}(z_i)$ which is the sum of the time it takes to survey each tile plus the flight time to get to and from the, presumably, closest home point.

With this formulation we can represent a feasible path as a tree subgraph of Q. The next step in the path planning process is to partition the path graph into a set of trees such that the each node (tile) in the graph is assigned to exactly one partition (path) (see Figures 5 and 6). We solve this problem two separate ways: with a breadth-first method and a MILP optimization problem. The breadth-first search is faster, since the MILP is a combinatorial optimization problem. However, the MILP produces a solution that minimizes the number of flights needed (Figure 6). Furthermore, the number of constraints in the MILP grows linearly with the number of nodes. This growth property is due to the fact that the path graph is inherently a DAG. If the graph was instead an undirected graph, finding tree-like partitions would require enumerating all possible self-loop constraints, which is exponential in the number of nodes.

Figure 7 shows each a set of solved tiles over West Cape Crozier and the entire unpartitioned path graph. Each tile represents a section of the survey to be completed. Partitioning the path graph

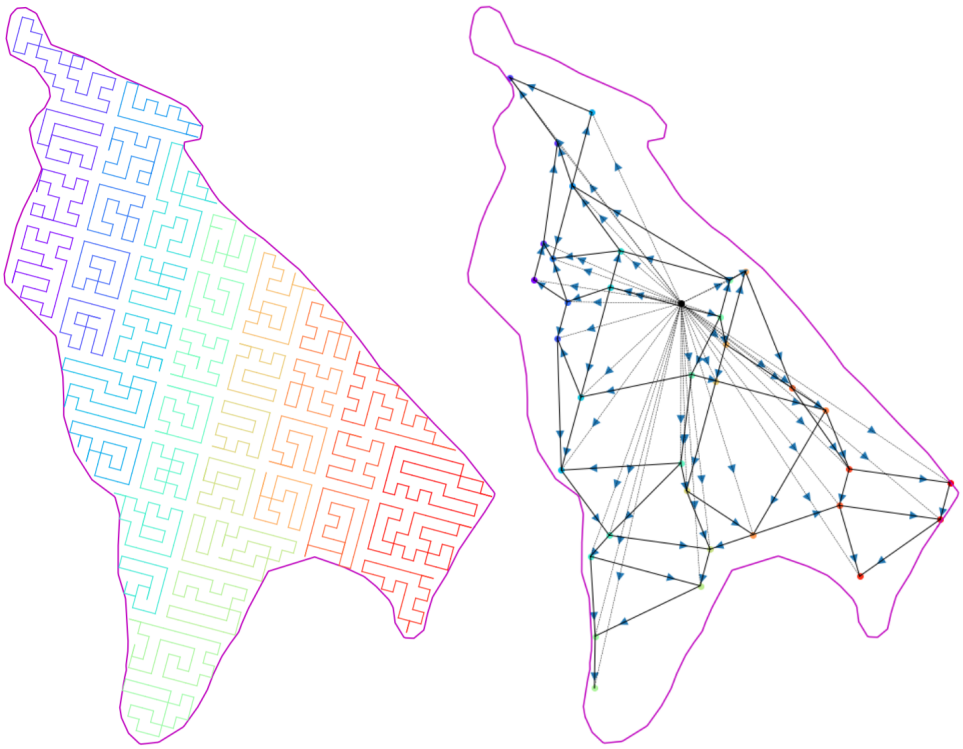

Figure 7. An example intermediate state of a set of solved tiles over the West Crozier geofence with the corresponding path graph. Each tile with corresponding path (left) with the complete unpartitioned path graph (right).

Algorithm 1. Breadth-First Partitioning

```
partitions = ∅
while nodes not empty do
    Tree = {}
    s ← MaxDist(nodes)
    Tree.addNode(s)
    Queue = {s}
    while Queue not empty do
        n ← Queue.pop()
        for p in predecessors(n) ∩ nodes do
            if cost(Tree + p) < limit then
                Tree.addNode(p)
                Queue.add(p)
                nodes.remove(p)
    partitions.append(tree)
```

allows us to effectively combine these sections in a scalable manner while balancing the transit and battery considerations.

Breadth-First Partitioning

We can generate a feasible partition by employing a breadth-first method to generate sets of trees. To build the partition, we first sort all the nodes in the path graph by their transit cost in descending order. Next we employ Algorithm 1 to extract the paths. This algorithm will take the farthest node and perform a breadth-first expansion which adds new nodes to the partition and checks if the total path is less than the specified limit.

The *cost()* method first attaches a home node to the closest (cheapest) node and then uses the edge weights to compute the total path cost based on the tiles in *Tree*. The algorithm has worst-case complexity of $\mathcal{O}(nd)$ where n is the number of nodes in the path graph and d is the maximum edge degree; however, due to the rectangular partitioning the maximum edge degree possible is 6 and in most cases is 4 for the majority of the tiles.

MILP Partitioning

We also formulate a MILP graph partitioning problem which produces a set of paths for a given value of K. We can iteratively lower K to minimize the total number of flights needed for a given survey. Since the path graph Q is a DAG, we can find subgraphs that are directed trees rooted at the home node. Each of these subgraphs is a partition of Q and contains a set of nodes (corresponding to tiles), such that the union of all partitions will contain all the nodes and thus no tile is ignored. Each subgraph is constrained to only contain one home edge with any number of base edges. To perform this partitioning, we can solve the corresponding optimization problem.

Let $x_1, \ldots, x_K \in \mathcal{B}^N$ and $z_1, \ldots, z_k \in \mathcal{B}^M$, where $\mathcal{B}^P$ is the set of binary variables $\{0, 1\}$ in P dimensions. Let N and M be the number of nodes and edges, respectively, of the path graph and K the number of desired partitions. For a given $k \in \{1, \ldots, K\}$, the elements of x_k and z_k that are 1 imply that those elements are in the kth partition. For example, if the third and fifth elements of x_k are 1 then nodes 3 and 5 are in partition k. All edges (home and base) have some weight $w_i \in R_+$ and represent the time needed to survey the corresponding tile. The goal of this formulation is to encode an assignment problem. For each k, the pair (x_k, z_k) represents an assignment of nodes and edges to the corresponding partition. Each of these partitions can be transformed into a path for a UAV to fly. While we need to assign every node in Q to a partition k, we need not assign every edge.

First we define the cost J, which is simply the total sum of all the edges taken,

$$J = \sum_{k=1}^{K} w^T z_k. \tag{12}$$

Next, we define a set of constraints that for a given k the variables x_k and z_k represent a tree. Since the path graph Q is a DAG we do not need to enforce costly self-loop constraints. We enforce that each node is assigned once except for the home node, which must be present in each partition. Without loss of generality we set the home node to the 0 index of x_k. We also constrain each edge to be present at least once,

$$\sum_{k=1}^{K} x_k = [K\ \mathbf{1}]^T, \tag{13}$$

$$\sum_{k=1}^{K} z_k \leq \mathbf{1}^T, \tag{14}$$

and that exactly one edge from the home node is assigned to each partition k,

$$\forall k\ \mathbf{1}_h^T z_k = 1, \tag{15}$$

where $\mathbf{1}_h$ is a vector of size m that is 1 for each edge that is directed away from the home node. The vector $\mathbf{1}_h$ encodes which of the m edges are the "home" edges. We require each partition total edge cost be less than the limit T_{limit} set by the UAV's flight time:

$$\forall k\ w^T z_k \leq T_{limit}. \tag{16}$$

Next we require that an edge can only exist in a partition k if both nodes are also in partition k,

$$2z_k[m] \leq x_k[n_1] + x_k[n_2]\ \forall k, \tag{17}$$

where m is the index of the directed edge $d = (n_1, n_2)$ from node n_1 to n_2. Lastly, we enforce a tree constraint where the number of nodes in the graph is exactly 1 less than the number of edges. This is a necessary and sufficient condition for a connected graph to be a tree (Diestel, 2004). Since each partition k represents a tree in a DAG, each (x_k, z_k) pair represents a directed tree:

$$\mathbf{1}^T x_k - 1 = \mathbf{1}^T z_k\ \forall k. \tag{18}$$

We see in Figures 8 and 9 that for the same problem instance (equal grid size and UAV parameters) the MILP partitioning method produces less flights than the breadth-first method. For this instance, 9 is the smallest possible number of flights since the MILP method iteratively decreases the allowable number of paths, K.

4. Performance

We define two important metrics, η_{path} and η_{total}, to measure the effectiveness of our method. Both these metrics along with total number of flights and total flight time are summarized in Table 1. η_{path} measures how much time is lost due to backtracking during the survey portion and η_{total} measures the amount of time lost due to transiting and backtracking. We show comparisons for a set of survey scenarios over the same area (West Cape Crozier) by increasing the coverage graph from 200 to 2,000 nodes which corresponds to a 60-m to 20-m step survey grid. We show our method's capability in not only reducing the number of flights needed but also the total survey time. We plot our comparison over three key performance metrics: number of flights, total flight time, and total route efficiency. Each planning instance ran on an i7-6700K CPU with 32GB of RAM and was given a maximum of 6 minutes to complete.

The metric

$$\eta_{path} = \frac{N_{path}}{N_{graph}}$$

is a measure of how much extra backtracking is incurred from relaxing the TSP and splitting the coverage graph with our tiling method, and is only applicable to our POPCORN+ SALT method.

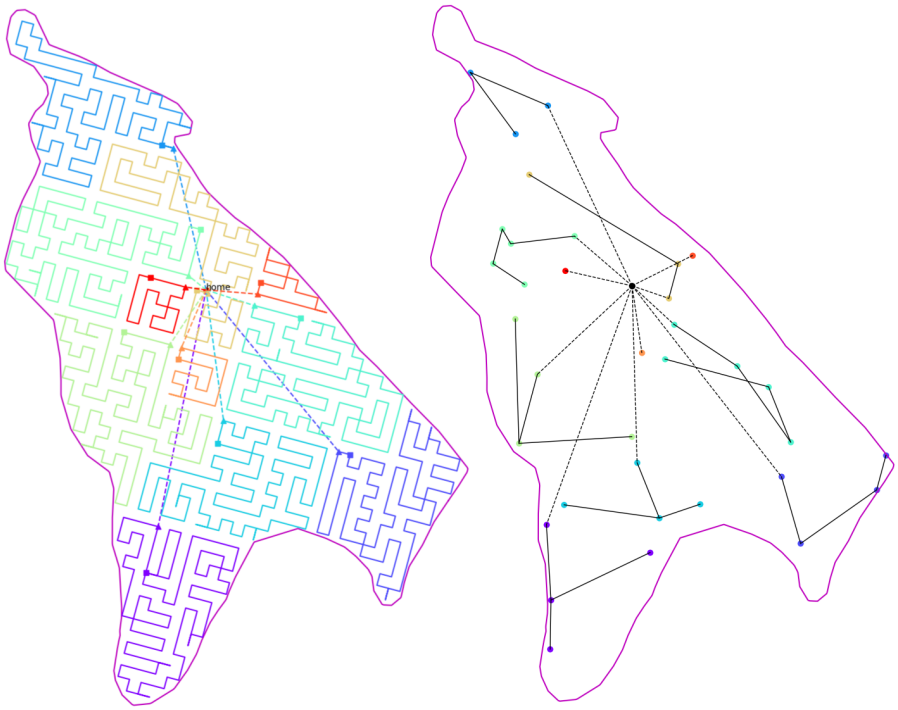

Figure 8. A completed planning instance using our breadth-first partitioning method over the Cape Crozier path using a 30-m step (left). The routes from the planner connected to the home location in the center with the path graph and corresponding partition (right). This shows 11 paths over Cape Crozier with the corresponding partitioned path graph.

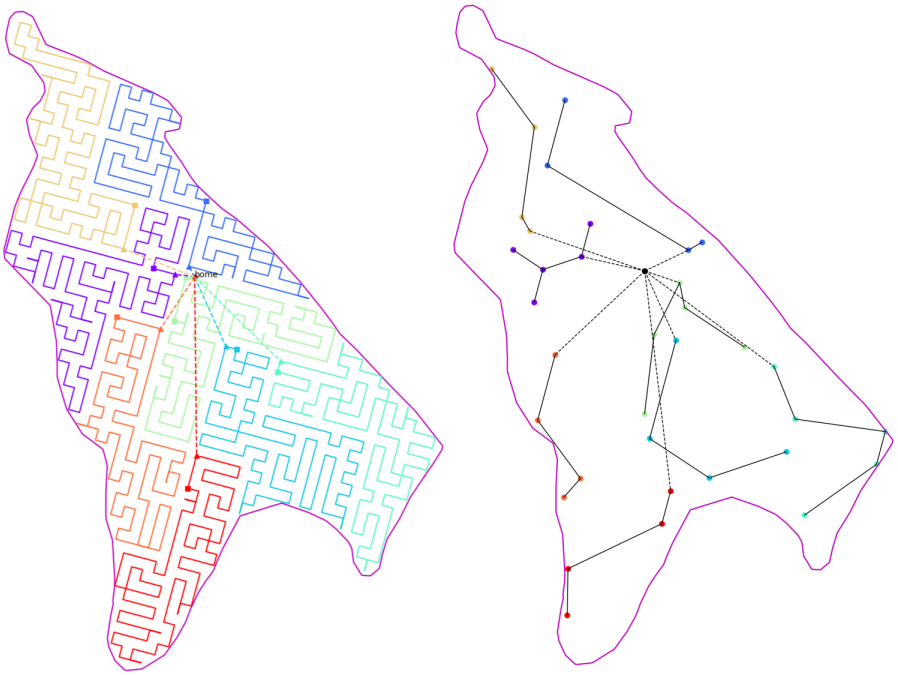

Figure 9. A completed planning instance using our MILP partitioning method over the Cape Crozier path using a 30-m step. The route from the planner connected to the home location in the center (left). The path graph with the corresponding partition (right). This shows nine paths over Cape Crozier with the corresponding partitioned path graph.

Table 1. Summary of results from different planning scenarios. We tabulate the path efficiency of our method as well as the total efficiency, number of flights, and the total flight time found by both of our methods (BFS, MILP) as well as the results from OR-Tools for the same problem instances. Since both of our methods (BFS and MILP) use the same subpaths (paths through each tile) they will have the same η_{path}. Further, η_{total} is the measure of how well these tiles are combined together to form the final flight paths.

Scenario		Ours	Ours: BFS			Ours: MILP			OR-Tools		
Grid (m)	Nodes	η_{path}	η_{total}	Flights	Time (min)	η_{total}	Flights	Time (min)	η_{total}	Flights	Time (min)
20	2275	0.920	0.829	17	175.877	0.860	14	170.147	0.771	17	192.388
21	2068	0.920	0.831	16	166.295	0.862	13	161.514	0.800	15	177.286
22	1887	0.924	0.837	15	158.867	0.866	12	153.554	0.802	14	168.895
23	1719	0.924	0.841	14	150.567	0.865	11	146.717	0.786	13	163.893
24	1589	0.919	0.838	13	144.505	0.859	11	142.007	0.817	12	152.296
25	1456	0.918	0.832	13	139.597	0.863	10	135.997	0.764	14	154.356
26	1348	0.920	0.829	13	134.222	0.856	10	132.363	0.790	12	144.163
27	1246	0.916	0.830	12	129.026	0.853	10	126.135	0.776	12	140.640
30	1011	0.922	0.836	11	115.214	0.860	9	113.461	0.782	11	125.213
31	946	0.916	0.837	10	111.676	0.856	9	108.912	0.758	11	124.844
32	894	0.911	0.823	11	110.731	0.849	9	107.272	0.755	11	121.983
33	837	0.914	0.833	10	105.520	0.850	8	103.763	0.758	10	117.452
34	788	0.922	0.827	11	101.690	0.869	8	98.065	0.760	10	113.456
35	742	0.912	0.836	9	96.890	0.855	8	96.863	0.756	10	110.299
36	705	0.900	0.825	9	97.582	0.842	8	96.387	0.740	10	109.797
37	661	0.917	0.839	8	91.265	0.850	8	92.228	0.740	9	106.088
38	628	0.910	0.817	9	92.877	0.838	8	89.950	0.728	9	105.118
39	597	0.916	0.829	9	88.988	0.840	8	88.720	0.827	9	90.074
40	564	0.913	0.845	8	83.512	0.857	7	82.603	0.792	9	90.618
41	540	0.911	0.832	8	84.773	0.841	7	83.195	0.770	8	91.919
42	515	0.918	0.847	7	79.950	0.849	7	79.505	0.813	8	84.779
43	488	0.914	0.830	8	78.922	0.855	6	77.829	0.781	8	85.465
44	471	0.915	0.833	7	77.912	0.845	7	77.912	0.855	7	77.325
45	450	0.915	0.828	8	77.420	0.835	7	76.210	0.782	8	82.184
46	429	0.911	0.836	7	75.555	0.842	7	75.000	0.781	8	79.969
47	409	0.905	0.817	7	73.912	0.841	6	71.968	0.782	8	77.539
48	399	0.913	0.834	7	71.845	0.851	6	70.740	0.817	8	73.758
49	380	0.916	0.839	7	67.549	0.842	7	70.372	0.794	7	74.196
51	347	0.916	0.826	6	67.308	0.853	5	64.936	0.778	7	71.544
52	337	0.908	0.811	7	67.594	0.824	6	66.497	0.782	7	70.471
53	326	0.924	0.839	6	65.146	0.844	6	62.632	0.777	7	69.731
54	311	0.909	0.832	6	62.744	0.837	6	63.170	0.840	7	62.383
55	302	0.904	0.832	6	64.161	0.832	6	63.531	0.804	7	64.430
56	288	0.906	0.824	6	58.061	0.825	6	59.235	0.788	7	63.721
57	282	0.910	0.803	7	63.264	0.818	6	62.134	0.769	7	64.838
58	267	0.914	0.799	7	57.311	0.804	7	60.929	0.827	6	58.480
59	261	0.913	0.801	7	59.396	0.822	6	58.496	0.817	6	58.812
60	251	0.909	0.818	6	57.624	0.821	6	57.829	0.812	6	57.568

This metric is the total number of steps, N_{path} (summed over all tiles), needed to route paths over a graph divided by the number of nodes in the graph, N_{graph}. The case where $\eta_{path} = 1.0$ occurs only when each tile is a Hamiltonian cycle, which is highly unlikely in practical routing scenarios. Table 1 shows that, for the suite of Crozier scenarios, $\eta_{path} \geq 0.9$ for all cases.

The metric

$$\eta_{total} = \frac{T_{unique}}{T_{total}}$$

is the total efficiency measured as the ratio between flight time over unique survey area, T_{unique}, and total flight time, T_{total}. T_{unique} can be calculated as

$$T_{unique} = \eta_{path} T_{survey},$$

where T_{survey} is the total time needed for the survey portion of the route. Total flight time T_{total} includes the time spent in transit plus the time spent surveying the area including any backtracking. This metric measures how efficient the route is at surveying/traversing over unique area; thus routes with short transit times and low backtracking will have a high total efficiency. Total flight time is simply the total amount that the agents must be in the air, which does not account for any ground time.

We compare our methods against the popular vehicle routing tool provided by Google OR-Tools (operations research) (Perron and Furnon, 2019). OR-Tools has many options for solving route planning problems. First it uses a set of search strategies to find an initial solution and then employs local search meta-heuristics to refine the solution. Since finding the optimal solution to the routing problem is computationally intractable, all of the OR-Tools options available are heuristic-based searches. For our comparisons we use the PATH CHEAPEST ARC option for the initial solution and the GUIDED LOCAL SEARCH option for the local search meta-heuristic.[5] Figure 12 shows that our MILP method produces the most efficient routes with an average total efficiency of 85%. The BFS method is slightly less efficient with an average of 2% less than the MILP method; both these methods are more efficient than the OR-Tools solutions by 6–8%.

As the coverage graph grows in size due to larger areas or grids spacing the number of flights needed to survey a given area also grows. These flights can either be done sequentially by a single drone or by dividing among a team of drones if available. Figure 10 shows how many flights it takes to completely survey for a varied grid size. All surveys were planned using a UAV with 13.5 minutes of flight time, which was the amount of flight time available for the DJI Matrice 100 used in our 2020 Antarctic field season. For reference, the 2019–2020 field season used a 38-m spacing and the 2020–2021 season used a 28-m spacing over west Cape Crozier. We see that the MILP method produces the lowest number of flights in all cases.

In addition to the increased number of flights, the total flight time also increases with increased coverage graph size. Figure 11 show that our methods produce shorter overall flight times in all planning instances. For coverage graphs larger than 600 nodes (38-m or smaller grid step), both the BFS and MILP methods produce total flight times an average of 15 minutes faster than the routes found by OR-Tools.

Google OR-Tools allows the user to set a maximum time to run the algorithm for; it will keep refining the solution until the time threshold is met. If this time is not set, it will keep running until a feasible solution is found. We compared our total computation time between both our methods (BFS and MILP). Since OR-Tools instead uses a time bound, we gave it a total of 6 minutes to find a path. Figure 13 shows a computation time comparison between our two methods, BFS and MILP, for a series of planning scenarios.

5. Case Studies

In the previous sections we outlined and described a novel multi-UAV path planning algorithm for use in aerial surveys. We specially designed the algorithm to be able to handle the constraints of real-world deployment scenarios. As evidence for readiness of our method, in this section we provide a series of detailed accounts of various survey case studies to better showcase the abilities and usefulness of our survey path planning methods. In the case of the Antarctic penguin surveys, We specifically compare the capabilities of SALT + POPCORN done in the 2020–2021 field season against a survey using POPCORN alone in the 2019–2020 field season. We also provide a case study

[5] developers.google.com/optimization/routing/tsp#search_strategy

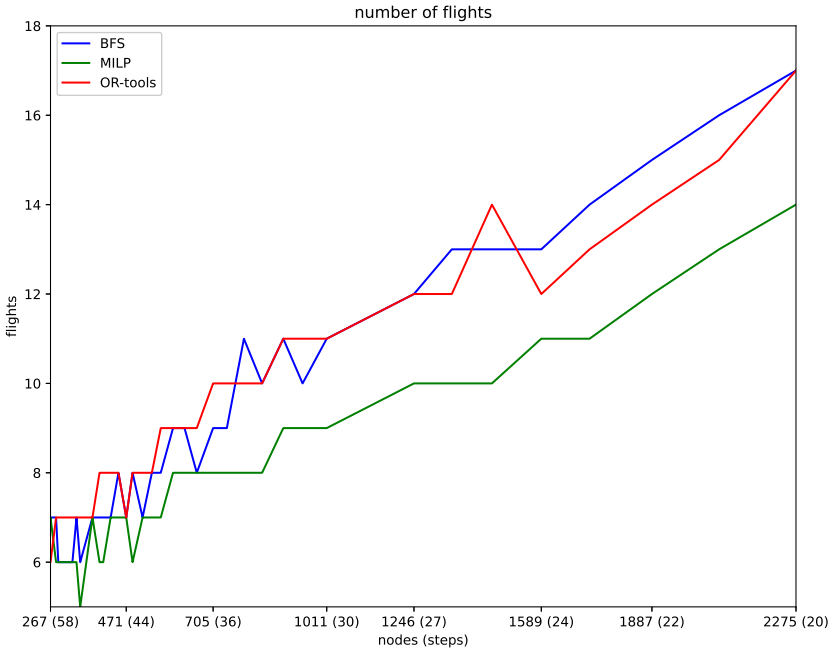

Figure 10. Number of flights to cover a fixed area vs coverage graph size where our methods BFS (blue) and MILP (green) are compared against Google OR-Tools (red). For a fixed survey area, as the grid size gets smaller the number of nodes grows quadratically. This figure shows that while our BFS method and Google's OR-Tools find roughly the same number of flights per planning instance, our MILP method is able to, by design, find solutions with less flights overall.

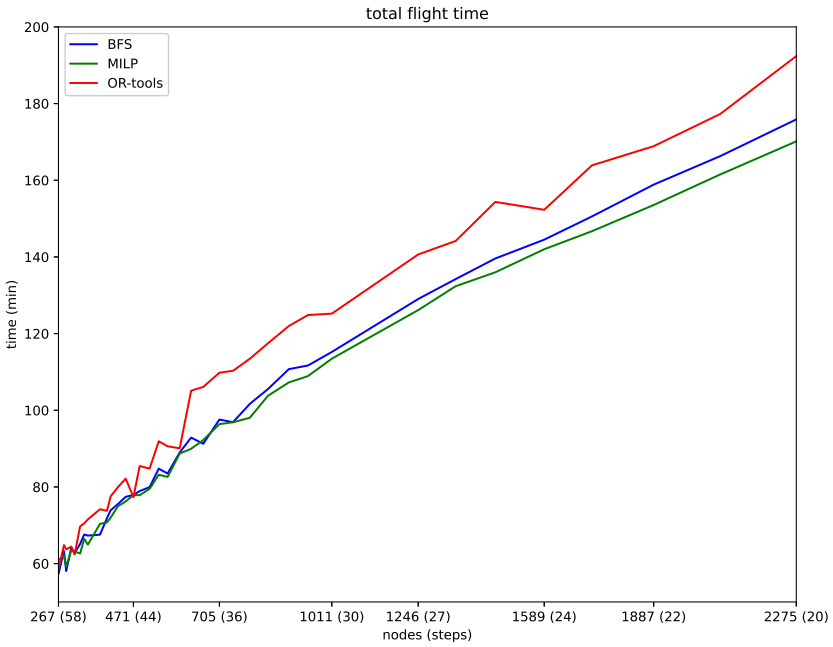

Figure 11. Total flight time vs coverage graph size. Our methods BFS (blue) and MILP (green) with Google's OR-Tools (red). This figure shows that our methods, on average, produce a set of routes that is, on average, 15 minutes faster than the Google OR-Tool paths.

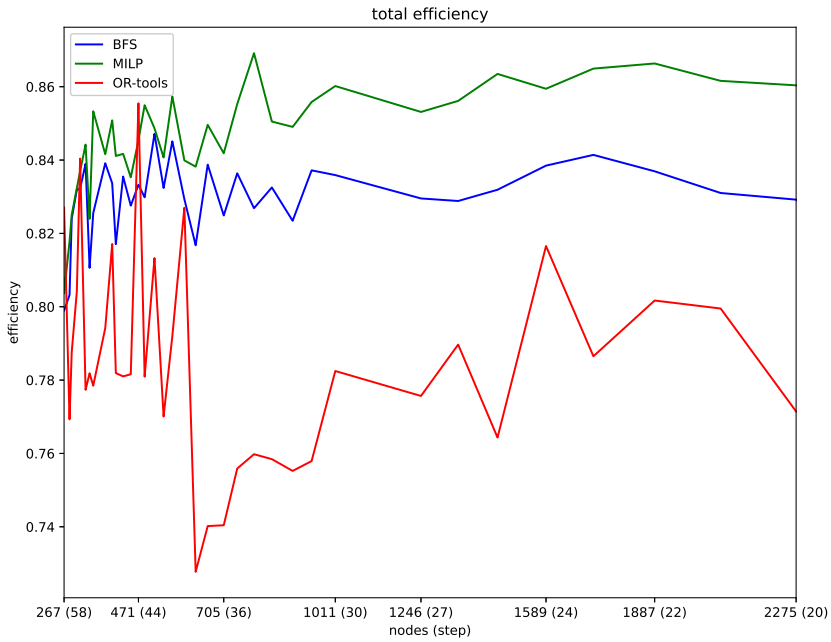

Figure 12. Total efficiency vs coverage graph size. Our methods BFS (blue) and MILP (green) with OR-Tools (red). The efficiency, η_{total}, measured here is a metric of how effective the planner is on not backtracking over the same area. This is calculated by dividing the amount of time spent surveying over the total flight time. We see that both our methods (BFS and MILP) are able to backtrack less overall than the method provided by Google OR-Tools.

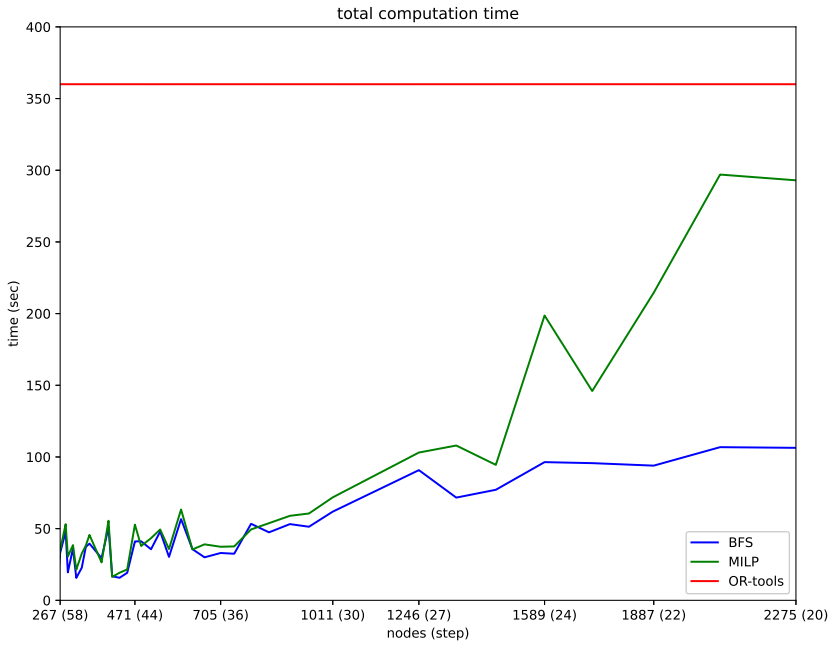

Figure 13. Total time vs coverage graph size. Our methods BFS (blue) and MILP (green) with OR-Tools (red). OR-Tools was given a bound of 360 seconds, where our methods were allowed to run till completion. For larger node sizes (small grid spacing) we see that the MILP method starts to rapidly grow in solution time, due to its exponential complexity whereas the BFS method grows at a much smaller rate.

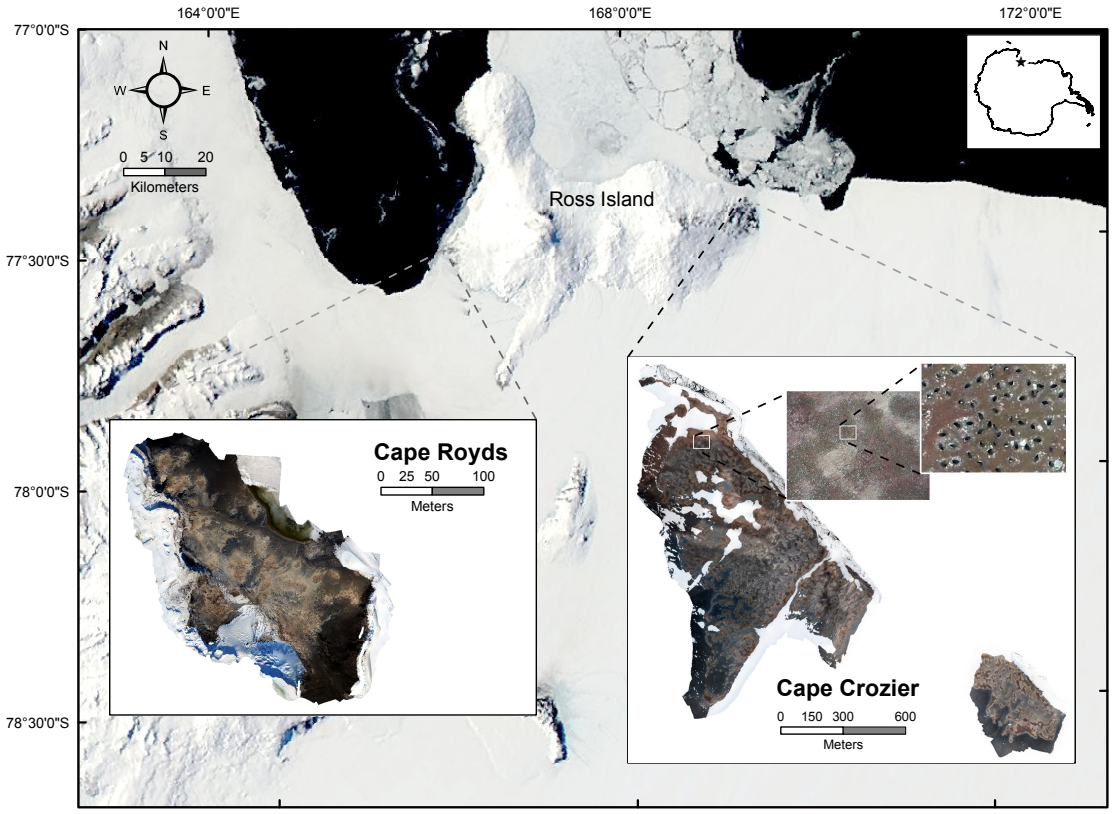

Figure 14. Cape Royds and Cape Crozier (including East Rookery) on Ross Island with location (star) on the Antarctic continent (top right). Full mosaics produced during our survey of Cape Crozier (bottom right) and Cape Royds (bottom left). An individual image from the Crozier survey with hundreds of nesting penguins expanded to show individual nests. Base satellite image of Ross Island vicinity acquired by the Moderate Resolution Imaging Spectroradiometer (MODIS) on board the Aqua satellite on 29 November 2011 (courtesy of Rapid Response Imagery from the Land, Atmosphere Near realtime Capability for Earth Observing System operated by the NASA/GSFC/Earth Science Data and Information System with funding provided by NASA/HQ).

of several islands in Mono Lake, Mono County, California, as well as a large ranch in Marin County, California.

5.1. Ross Island, Antarctica

The Adélie penguin (*Pygoscelis adeliae*) colony at Cape Crozier is situated at roughly sea level to 100 m above median sea level (AMSL) on the coast of Ross Island. Between the colony and a small five-person field camp there are rocky hills (100–400 m AMSL), ice fields, and nesting South Polar Skuas (*Stercorarius maccormick*). These geographic constraints as well as penguins moving about the colony, windy weather, and rocky snowy terrain made it difficult to establish safe take-off and landing zones within or around the edge of the colony. Furthermore, keeping batteries warm enough for flight (15°C) in the face of subzero conditions (temperatures over the austral summer season are −15°C to 0°C) for hours at a time was infeasible in the field. These restrictions required the UAVs to take off at the field camp, enter the colony (2 km from the camp) at key safe points, decelerate to a nondisruptive speed to take survey photos, and finally exit the colony near or at the same key safe point to accelerate and fly back to the field camp. While the chief focus of our surveys was the Cape Crozier colony (Figure 14), the largest colony with over 300,000 nests spreading over an area of 2.5 km^2, including a distinct section ("East Rookery") with ∼30,000 nests, we also surveyed

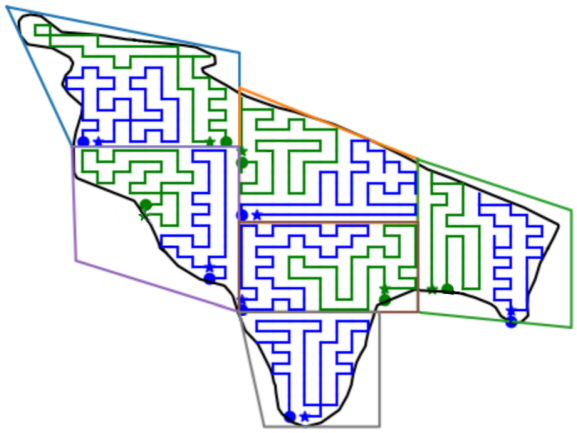

Figure 15. Cape Crozier flight plans for the 2019–2020 season. Six manually created tiles were found and solved as single path planning instances using just POPCORN.

a small colony at Cape Royds with ∼2,500 nests (see Figure 14). Specifically we show that SALT provided a massive improvement to flexibility and speed over just POPCORN.

2019–2020. Between November 2019 and February 2020, we fielded a team of four UAVs executing paths planned with only POPCORN to conduct a total of 11 aerial surveys of multiple Adélie penguin colonies on Ross Island, Antarctica. Due to the size of the survey area we manually divided the main Crozier area into six tiles (z0–z5), seen as colored polygons in Figure 15. The main Cape Crozier colony required 11 flights and the adjacent East Rookery required 2 flights. Thus the total number of flights over the whole Crozier colony was 13 per survey. More details can be found in Shah et al. (2020). Total computation for these routes took over 50 hours.

2020–2021. Due to the COVID-19 pandemic, field personnel were limited for the 2020–2021 Antarctic field season. The Cape Crozier field camp had very limited staff (two people) for the duration of the whole field season and only two UAVs could be flown for each survey. As a result, the overall field flight plan was changed to accommodate these constraints. Notably the main Crozier field site was split into four distinct geofences and the UAVs were launched from within the colony instead of from the field camp as was done in the previous field season. With the addition of SALT to POPCORN the field team was able to rapidly replan in the field when needed as the initial take-off locations were unknown prior to arrival. Figure 16 shows the paths imported into (SPH Engineering, n.d.) and Figure 17 shows the planning output from `wadl` over the four separate sections of Cape Crozier using the MILP stitching method. The grid spacing is lower from the previous year (28-m grid down from 40 m). Since the UAVs do not need to transit as much to get to the colony, only 11 flights were needed for the main Crozier area with 2 additional flights for the East Rook area. Total computation time for the entire survey was less than 5 minutes using SALT + POPCORN, as compared to the previous year's survey which took over 50 hours to solve. A survey planned with both our methods (BFS and MILP) using a single geofence as well as total computation times can be found in Section 3.

5.2. Mono Lake

Mono Lake (Figure 18) in eastern California is a large hypersaline lake, and home to one of the largest breeding colonies of California gulls (*Larus californicus*) in the world. The gull population is used as an indicator to guide management of the lake ecosystem (Burnett et al., 2021). The gulls nest primarily on a series of islands located within an approximately 14 km^2 area in the north-central

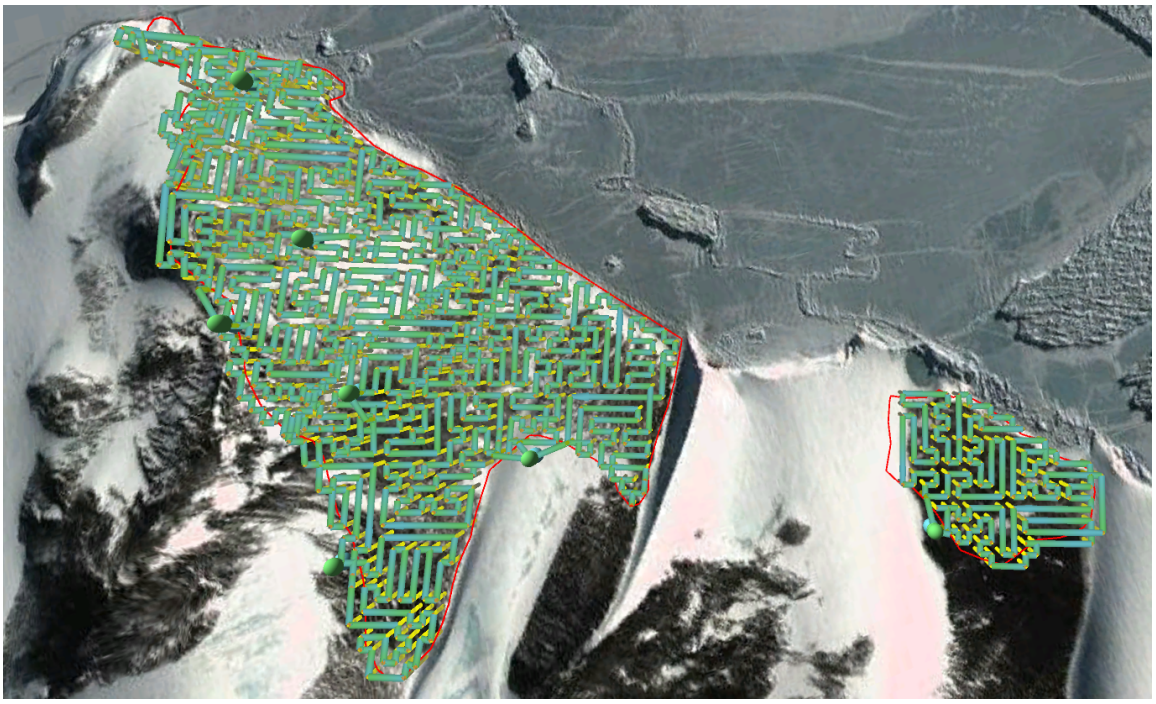

Figure 16. Routes for the 2021 field season imported into UGCS and ready for deployment.

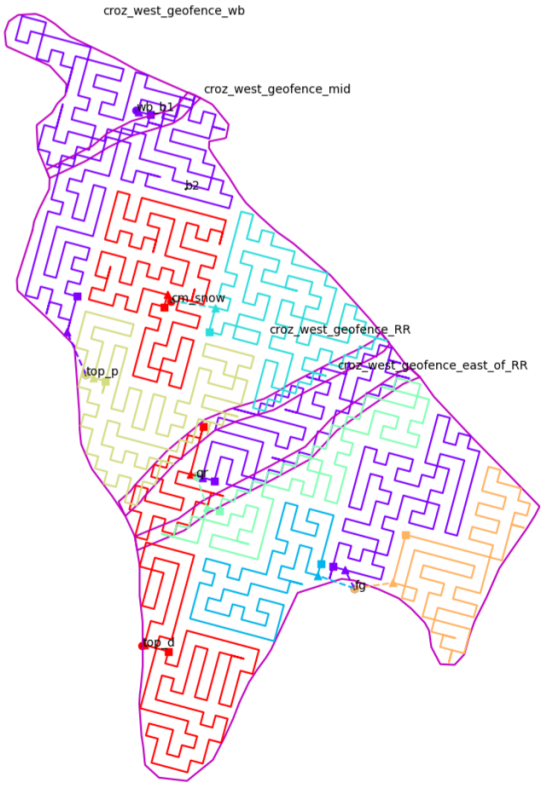

Figure 17. WADL path output using the MILP method over Cape Crozier.

Figure 18. Mono Lake. UAV and a California gull can also be seen.

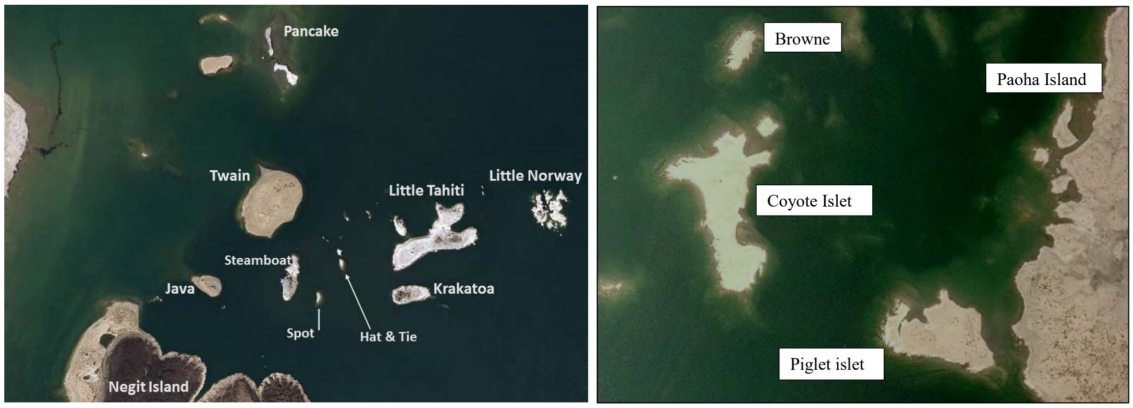

Figure 19. The Negit (left) and Paoha (right) islands and islets. All named islands (except Paoha) were surveyed.

portion of the lake. Many of the gulls' nests can be found on the Negit (103 ha) and Paoha (810 ha) islands, as well as the smaller surrounding islands known as the Negit and Paoha islets. Figure 19 show all the islands surveyed (images taken from Burnett et al., 2021).

Figure 20 shows a flight plan over the Negit islets. Routes over smaller islands (e.g., Spot, Tie, Hat) were combined together for better battery utilization. Figure 21 shows two UAVs over Steamboat and Little Tahiti during the May 2020 survey.

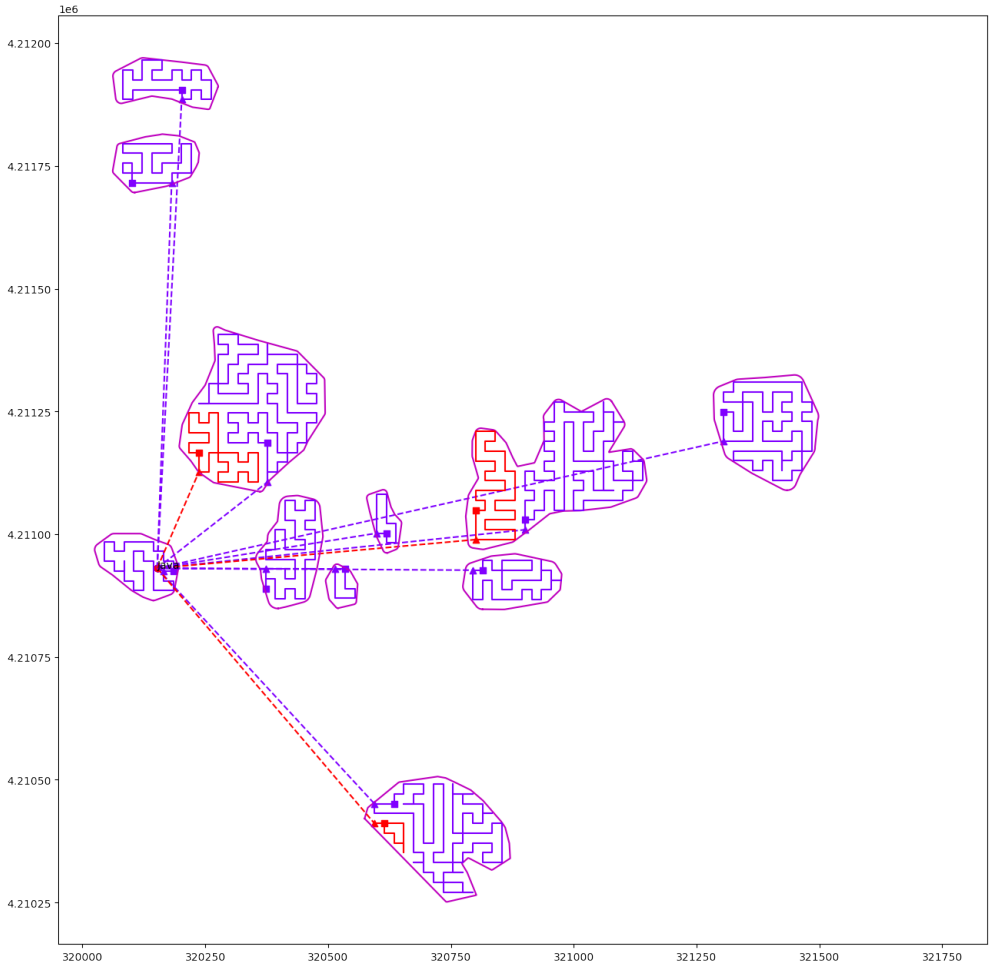

Figure 20. A set of routes planned for the Negit islets. Some routes were connected together for better battery utilization. Units are in Universal Transverse Mercator (UTM) and this figure spans about 2×2 km^2.

In both 2020 and 2021 the gull population was surveyed twice, once before the egg hatching period (late May) and once just prior to the chicks leaving (fledging) (mid-July). These two surveys were then compared to determine the breeding success (the number of chicks produced per breeding pair) of the gulls. These population data were also used to compare against the previous year's measurements to better understand the trends of the gull population.

5.3. Marin Ranch

We surveyed a ranch in Marin County, just north of San Francisco, California. The ranch is over 2000 acres and can be seen in Figure 22. Due to line-of-sight constraints as well as UAV range constraints we divided the ranch into three sections, where the first section took approximately 4 hours to survey at 70 m AGL using four UAVs and the last two sections took approximately 2 hours each to survey. Figure 23 shows the different routes over each section of the ranch and the final orthomosaic from all the stitched images.

6. Conclusion

This work provides a general and flexible path planning tool for a team of aerial UAVs to complete aerial coverage tasks. The work culminates in a survey planning tool for any number of UAVs that

Figure 21. Two UAVs surveying a set of islands over Negit islets.

Figure 22. A UAV taking off over the ranch overlooking some cattle.

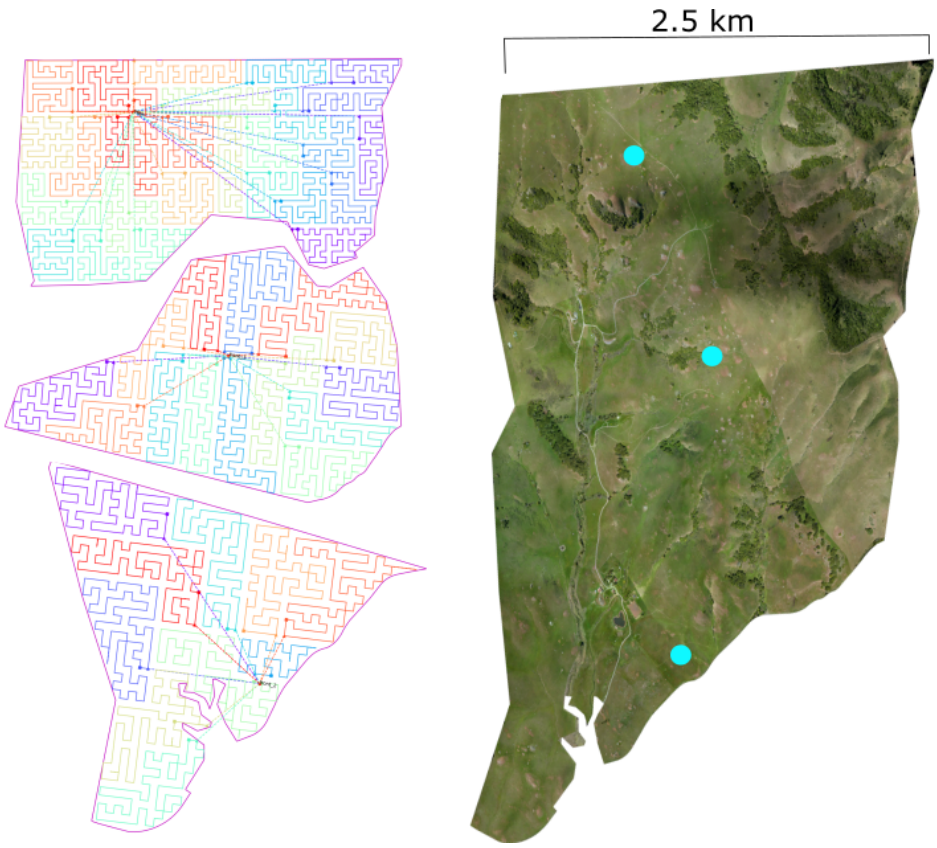

Figure 23. Left: Planned routes over the Marin, California, ranch. Each of the three sections is shown with all the corresponding routes: the northern-most section of the ranch with 33 flight routes, the middle section of the ranch with 13 routes, and the southern section of the ranch with 10 routes. Right: The final orthomosaic of the ranch, stitched together from over 16,000 separate images. The home launch/land location are also marked as cyan circles.

has the capability for online replanning to adapt in real time to changing survey conditions, as well as a method for decentralized in-flight collision avoidance. Our method reduces survey time by limiting redundant travel while also allowing for safe recall of the drones at any time during the survey. Our tool addresses the need for a system that can generate paths that are not only efficient but also respect the limited battery life of the UAVs as well as any other operational constraints such as take-off and landing locations. We describe extensive field deployments of our system in a variety of landscapes, including icy and rocky terrain, lake regions, and large grassland areas.

Lessons Learned and Future Work

The route planning method described in this paper has afforded numerous surveys of different habitats across the globe. However, some difficulties still arise during these on-site deployments. In this section we outline some lessons learned from the various field teams that have used this system in their data gathering efforts. We also list some potential areas for future work.

Elevation Models

Digital elevation models (DEMs) are used extensively in aerial survey operations as they contain vital altitude information for the terrain being surveyed. In poorly surveyed areas, or areas with heavy vegetation, the DEM may contain considerable error. In these cases we found that the UAVs would either climb erroneously high over the area, or the vegetation height would be underestimated

altogether and increase the risk for collision. To address these issues we plan to integrate RBG-D cameras into our UAVs to better estimate altitude relative to other collision risks.

Tile Generation

The addition of SALT introduced new meta-parameters such as tile size, as well as minimum and maximum path lengths for each tile. We are exploring preprocessing methods that look at the connectivity structure of each tile to better estimate a lower bound and to better seed the SAT solving. In some cases, like areas with narrow passages, there are linear graph-like sections that would imply a tile is a better lower bound than the one estimated from the length of the potential Hamiltonian cycle. Furthermore, during routing we have noticed that occasionally the paths contain undesirable geometries which can cause inconsistencies during flight. For example, sometimes the route contains a long section in which the UAV is meant to take a 180° turn. This can cause the image to not be taken at the correct location because of how the UAV's acceleration profile controls it through the turn. We are exploring ways to detect certain undesirable patterns before routing such that we can partially merge tiles in a way that avoids these types of artifacts.

In Flight Failures

Occasionally in our field deployments, we needed to recall a UAV from the field. These recalls were due to either human error (e.g., failure to swap out a full memory card from the camera) or environmental issues such as sudden weather changes, and often resulted in a partially completed route. Other equipment issues such as faster than expected battery drain or loss of compass calibration also led to recalls during operation. We are exploring methods that would be able to reassign the incomplete points back into the remaining routes, or generate a new route while still minimizing the overall flight time. Some preliminary methods allow the planner to redistribute waypoints to other routes in the case of mid-survey recalls. This is achieved by solving a task-assignment-like problem that splits and merges the remaining routes to rebalance the load of the points not yet surveyed on the failed route. In some cases no extra routes need to be flown. We plan to test this and other features in subsequent deployments.

Acknowledgments

We thank D. Ainley, R. Burnett, C. Carey, R. DiGaudio, G. Halstrom, K. Dugger, M. Elrod, D. Jongsomjit, E. Porzig, P. Levinson, A. Lescroël, and V. Morandini for assistance with the field operations. This work was supported by NSF grant 1834986 with logistical support in Antarctica provided by the United States Antarctic Program. We are grateful to the Mono Lake Committee for their continued partnership and for providing financial and logistical support for the annual California gull survey. We thank California State Park staff from the Mono Lake Tufa State reserve and the Inyo National Forest for their support and collaboration as well. We also thank an anonymous donor for their support and collaboration on the ranch survey. An open-source version of the codebase for this project can be found at github.com/k2shah/wadl. We would like to thank every Adélie penguin on Ross island for their patience and being so photogenic.

ORCID

Kunal Shah https://orcid.org/0000-0003-1550-4546
Annie E. Schmidt https://orcid.org/0000-0001-6144-9950
Grant Ballard https://orcid.org/0000-0002-8604-7066
Mac Schwager https://orcid.org/0000-0002-7871-3663

References

Acar, E. U., Choset, H., Rizzi, A. A., Atkar, P. N., and Hull, D. (2002). Morse decompositions for coverage tasks. *International Journal of Robotics Research*, 21(4):331–344.

Agisoft (2019). Metashape. agisoft.com.

Agmon, N., Hazon, N., and Kaminka, G. A. (2006). Constructing spanning trees for efficient multi-robot coverage. In *Proceedings 2006 IEEE International Conference on Robotics and Automation (ICRA 2006)*, pages 1698–1703. IEEE.

Almadhoun, R., Taha, T., Seneviratne, L., and Zweiri, Y. (2019). A survey on multi-robot coverage path planning for model reconstruction and mapping. *SN Applied Sciences*, 1(8):1–24.

Andreev, K. and Racke, H. (2006). Balanced graph partitioning. *Theory of Computing Systems*, 39(6):929–939.

Azpúrua, H., Freitas, G. M., Macharet, D. G., and Campos, M. F. (2018). Multi-robot coverage path planning using hexagonal segmentation for geophysical surveys. *Robotica*, 36(8):1144–1166.

Bähnemann, R., Lawrance, N., Chung, J. J., Pantic, M., Siegwart, R., and Nieto, J. (2021). Revisiting boustrophedon coverage path planning as a generalized traveling salesman problem. In *Field and Service Robotics*, pages 277–290. Springer.

Bektas, T. (2006). The multiple traveling salesman problem: An overview of formulations and solution procedures. *Omega*, 34(3):209–219.

Burnett, R. D., Nelson, K., and Schmidt, A. E. (2021). Population size and reproductive success of California gulls at Mono Lake, California.

Cabreira, T., Brisolara, L., and Ferreira, P. R., Jr. (2019a). Survey on coverage path planning with unmanned aerial vehicles. *Drones*, 3(1):4.

Cabreira, T. M., Ferreira, P. R., Franco, C. D., and Buttazzo, G. C. (2019b). Grid-based coverage path planning with minimum energy over irregular-shaped areas with UAVs. In *2019 International Conference on Unmanned Aircraft Systems (ICUAS)*, pages 758–767.

Choset, H. (2001). Coverage for robotics—A survey of recent results. *Annals of Mathematics and Artificial Intelligence*, 31(1–4):113–126.

Choset, H. and Pignon, P. (1998). Coverage path planning: The boustrophedon cellular decomposition. In *Field and service robotics*, pages 203–209. Springer.

Choset, H., Acar, E., Rizzi, A. A., and Luntz, J. (2000). Exact cellular decompositions in terms of critical points of morse functions. In *Proceedings 2000 ICRA, Millennium Conference, IEEE International Conference on Robotics and Automation, Symposia Proceedings (Cat. No. 00CH37065)*, volume 3, pages 2270–2277. IEEE.

De Carvalho, R. N., Vidal, H., Vieira, P., and Ribeiro, M. (1997). Complete coverage path planning and guidance for cleaning robots. In *ISIE'97 Proceedings of the IEEE International Symposium on Industrial Electronics*, volume 2, pages 677–682. IEEE.

De Moura, L. and Bjørner, N. (2008). Z3: An efficient SMT solver. In *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, pages 337–340. Springer.

Di Franco, C. and Buttazzo, G. (2015). Energy-aware coverage path planning of UAVs. In *Proceedings 2015 IEEE International Conference on Autonomous Robot Systems and Competitions (ICARSC 2015)*, pages 111–117. IEEE.

Diestel, R. (2004). *Graph Theory*, 5th edition. Springer.

DroneDeploy (2021). Drone deploy. https://www.dronedeploy.com.

Englot, B. and Hover, F. (2012). Sampling-based coverage path planning for inspection of complex structures. In *Proceedings of the International Conference on Automated Planning and Scheduling*, volume 22.

Gabriely, Y. and Rimon, E. (2001). Spanning-tree based coverage of continuous areas by a mobile robot. In *Proceedings 2001 ICRA, IEEE International Conference on Robotics and Automation (Cat. No.01CH37164)*, volume 2, pages 1927–1933.

Galceran, E. and Carreras, M. (2013). A survey on coverage path planning for robotics. *Robotics and Autonomous Systems*, 61(12):1258–1276.

Gutin, G. and Yeo, A. (2007). The greedy algorithm for the symmetric TSP. *Algorithmic Operations Research*, 2(1).

Held, M. and Karp, R. M. (1970). The traveling-salesman problem and minimum spanning trees. *Operations Research*, 18(6):1138–1162.

Held, M. and Karp, R. M. (1971). The traveling-salesman problem and minimum spanning trees: Part II. *Mathematical Programming*, 1(1):6–25.

Huang, W. H. (2001). Optimal line-sweep-based decompositions for coverage algorithms. In *Proceedings 2001 ICRA, IEEE International Conference on Robotics and Automation (Cat. No. 01CH37164)*, volume 1, pages 27–32. IEEE.

Hung, W. N. N., Song, X., Tan, J., Li, X., Zhang, J., Wang, R., and Gao, P. (2014). Motion planning with satisfiability modulo theories. In *2014 IEEE International Conference on Robotics and Automation (ICRA)*, pages 113–118.

Imeson, F. and Smith, S. L. (2019). An SMT-based approach to motion planning for multiple robots with complex constraints. *IEEE Transactions on Robotics*, 35(3):669–684.

Jing, W., Deng, D., Wu, Y., and Shimada, K. (2020). Multi-UAV coverage path planning for the inspection of large and complex structures. In *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 1480–1486.

Jovanovic, R. and Voß, S. (2016). A mixed integer program for partitioning graphs with supply and demand emphasizing sparse graphs. *Optimization Letters*, 10(8):1693–1703.

Kapanoglu, M., Alikalfa, M., Ozkan, M., Yazıcı, A., and Parlaktuna, O. (2012). A pattern-based genetic algorithm for multi-robot coverage path planning minimizing completion time. *Journal of Intelligent Manufacturing*, 23(4):1035–1045.

Kapoutsis, A. C., Chatzichristofis, S. A., and Kosmatopoulos, E. B. (2017). DARP: Divide areas algorithm for optimal multi-robot coverage path planning. *Journal of Intelligent & Robotic Systems*, 86(3–4):663–680.

Kernighan, B. W. and Lin, S. (1970). An efficient heuristic procedure for partitioning graphs. *The Bell System Technical Journal*, 49(2):291–307.

Kong, C. S., Peng, N. A., and Rekleitis, I. (2006). Distributed coverage with multi-robot system. In *Proceedings 2006 IEEE International Conference on Robotics and Automation (ICRA 2006)*, pages 2423–2429. IEEE.

Microsoft. Image composite editor. microsoft.com/en-us/research/project/image-composite-editor/.

Moon, S.-W. and Shim, D. H.-C. (2009). Study on path planning algorithms for unmanned agricultural helicopters in complex environment. *International Journal of Aeronautical and Space Sciences*, 10(2):1–11.

Nam, L., Huang, L., Li, X. J., and Xu, J. (2016). An approach for coverage path planning for UAVs. In *2016 IEEE 14th International Workshop on Advanced Motion Control (AMC)*, pages 411–416. IEEE.

Oksanen, T. and Visala, A. (2009). Coverage path planning algorithms for agricultural field machines. *Journal of Field Robotics*, 26(8):651–668.

Otto, A., Agatz, N., Campbell, J., Golden, B., and Pesch, E. (2018). *Optimization Approaches for Civil Applications of Unmanned Aerial Vehicles (UAVs) or Aerial Drones: A Survey*, volume 72. Wiley-Liss.

Perron, L. and Furnon, V. (2019). OR-Tools. https://developers.google.com/optimization/.

PIX4D Inc. PIX4Dcapture. https://www.pix4d.com/product/pix4dcapture.

Scala, E., Ramirez, M., Haslum, P., and Thiebaux, S. (2016). Numeric planning with disjunctive global constraints via SMT. In *26th International Conference on Automated Planning and Scheduling*.

Shah, K., Ballard, G., Schmidt, A., and Schwager, M. (2020). Multidrone aerial surveys of penguin colonies in Antarctica. *Science Robotics*, 5(47).

Shoukry, Y., Nuzzo, P., Saha, I., Sangiovanni-Vincentelli, A. L., Seshia, S. A., Pappas, G. J., and Tabuada, P. (2016). Scalable lazy SMT-based motion planning. In *2016 IEEE 55th Conference on Decision and Control (CDC)*, pages 6683–6688. IEEE.

SPH Engineering. Universal Ground Control Station. ugcs.com.

Surynek, P. (2014). Simple direct propositional encoding of cooperative path finding simplified yet more. In *Mexican International Conference on Artificial Intelligence*, pages 410–425. Springer.

Surynek, P. (2015). Reduced time-expansion graphs and goal decomposition for solving cooperative path finding sub-optimally. In *Proceedings 24th International Joint Conference on Artificial Intelligence (IJCAI 2015)*, pages 1916–1922.

Toth, P. and Vigo, D. (2002). *The Vehicle Routing Problem*. SIAM.

Zheng, X., Jain, S., Koenig, S., and Kempe, D. (2005). Multi-robot forest coverage. In *2005 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 3852–3857. IEEE.

How to cite this article: Shah, K., Schmidt, A. E., Ballard, G., & Schwager, M. (2022). Large scale aerial multi-robot coverage path planning. *Field Robotics, 2*, 1971–1998.