Line Coverage with Multiple Robots: Algorithms and Experiments

Saurav Agarwal and Srinivas Akella

Abstract—The line coverage problem involves finding efficient routes for the coverage of linear features by one or more resource-constrained robots. Linear features model environments such as road networks, power lines, and oil and gas pipelines. Two modes of travel are defined for robots: servicing and deadheading. A robot services a feature if it performs task-specific actions, such as taking images, as it traverses the feature; otherwise, it is deadheading. Traversing the environment incurs costs (e.g., travel time) and resource demands (e.g., battery life). Servicing and deadheading can have different cost and demand functions, which can be direction dependent. The environment is modeled as a graph, and an integer linear program is provided. As the problem is NP-hard, we design a fast and efficient heuristic algorithm, Merge-Embed-Merge (MEM). Exploiting the constructive property of the MEM algorithm, algorithms for line coverage of large graphs with multiple depots are developed. Furthermore, turning costs and nonholonomic constraints are efficiently incorporated into the algorithm. The algorithms are benchmarked on 100 road networks and demonstrated in experiments with aerial robots.

Index Terms—Path Planning for Multiple Mobile Robots or Agents, Aerial Systems: Applications, Motion and Path Planning, Arc Routing Problems

I. INTRODUCTION

LINE coverage is the task of servicing linear environment features using sensors or tools mounted on robots. The features to be serviced are modeled as one-dimensional segments (or curves), and all points along the segments must be visited. Consider a natural disaster scenario such as flooding. A team of uncrewed aerial vehicles (UAVs) is deployed to assess the accessibility of a road network for emergency services. The UAVs must traverse the line segments corresponding to the road network and capture images for analysis. This paper seeks to answer the following question: How should efficient routes for UAVs be planned so that each road network segment is serviced? Mobile robots are often resource-constrained—they must come back to their *depot* or launch location before they exhaust their resources, such as battery life. Figure 1 shows a large road network, routes for eight UAVs covering the entire road network, and an orthomosaic generated from the images taken by the team of UAVs. Power lines and gas

Saurav Agarwal is with the GRASP Lab at the University of Pennsylvania and Srinivas Akella is with the Department of Computer Science, University of North Carolina at Charlotte, NC 28223, USA, where this work was performed. Corresponding author: Saurav Agarwal.
E-mail: `sauravag@seas.upenn.edu`, `sakella@charlotte.edu`

pipelines have similar infrastructure that requires frequent inspection. Additional applications arise in perimeter inspection and surveillance, traffic analysis of road networks, and welding and 3D printing operations.

Line coverage is closely related to arc routing problems (ARPs) studied in the operations research community [1]. ARPs have been used to generate routes for snow plowing, spraying salt, and cleaning road networks [2]. The ARPs and their algorithms are designed specifically for human-operated vehicles. Although the above tasks can potentially be automated with ground robots, line coverage has received limited attention in the robotics community. In this paper, we design algorithms for line coverage using autonomous systems applicable to aerial, ground, and underwater robots.

The line coverage problem with multiple robots, modeled using a graph, has three defining attributes: (1) The edges in the graph are classified as required and non-required, (2) Robots have two modes of travel—servicing and deadheading, and (3) Robots have constraints on the resources available to them. Required edges in the graph correspond to the line features to be covered, and a robot can use the non-required edges to travel from one vertex to another to reduce cost. The vertices in the graph represent the endpoints of the edges. A robot is said to be *servicing* a required edge when it performs task-specific actions such as collecting sensor data. Each required edge needs to be serviced exactly once by any robot. Robots may also traverse an edge without performing servicing tasks to optimize travel time, conserve energy, or reduce the amount of sensor data. This mode of travel is known as *deadheading*, and both types of edges may be used any number of times for this purpose.

A service cost and a deadhead cost (e.g., travel time) are associated with each required edge, and they are incurred each time an edge is serviced or deadheaded, respectively. Only the deadhead cost is associated with the non-required edges. The total sum of the service and deadhead costs over all routes is to be minimized. Moreover, traversing an edge results in the consumption of resources such as battery life. These are modeled as *demands* on the edges, and the total demand of a route should be less than the given *resource capacity* of the robots. The costs and demands for servicing an edge are usually more than those for deadheading, as task-related actions are only performed while servicing. For example, a robot servicing an edge by recording images may travel slower to avoid motion blur, resulting in a longer travel time.

The *line coverage problem with multiple robots* is the task of computing efficient coverage routes for a set of line features such that the total cost of travel is minimized while respecting

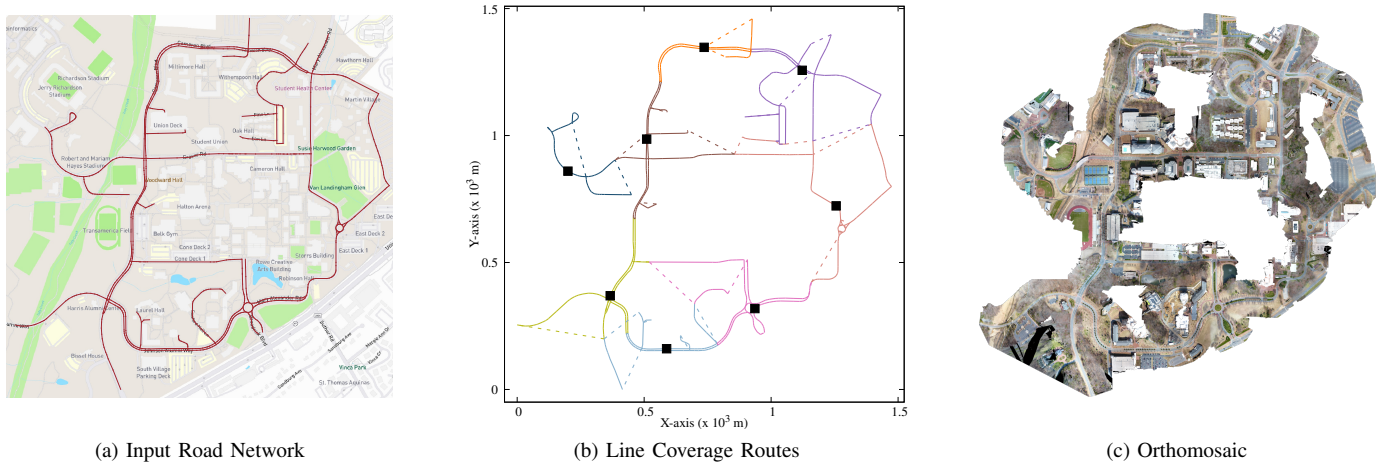(a) Input Road Network   (b) Line Coverage Routes   (c) Orthomosaic

Fig. 1. Line coverage of the UNC Charlotte road network using a team of resource-constrained UAVs. (a) The input road network is 13 km in length and spans an area of 1.5 km². The road network is modeled as a graph comprising 842 vertices and 865 required edges representing the road segments. As the UAVs can fly from one vertex to another, we add a non-required edge between each pair of vertices, resulting in 353,196 non-required edges in the graph. (b) Eight routes for a team of UAVs are computed using the multi-depot Merge-Embed-Merge (MEM) algorithm developed in this paper. The algorithm computes depot locations, shown by black squares, from where the UAVs start and end their routes. The solid lines represent servicing, while the dashed lines represent deadheading. The UAVs can fly faster while deadheading, thereby optimizing total flight time. (c) An orthomosaic of the road network generated from the images taken by the UAVs flown autonomously along the computed routes.

the resource constraints. Starting with the line coverage problem for asymmetric graphs as the core problem, we formulate an integer linear program (ILP) and design a constructive heuristic algorithm called Merge-Embed-Merge (MEM). To generate deployable solutions for the line coverage problem in real-world scenarios, we address three important practical aspects: (1) Asymmetry in costs and demands, (2) Large graphs that require multiple depots, and (3) Turning costs and nonholonomic constraints. This paper, for the first time, models these important practical aspects and presents a unified approach for the line coverage problem by developing highly efficient algorithms to compute routes for the robots.

Asymmetry in costs and demands: In many robotics applications, the cost of travel and resource demands are direction-dependent. For example, a ground robot traveling uphill can take longer and consume more energy than when traveling downhill. Similarly, the costs and demands of aerial robots may differ in two directions due to wind conditions. Hence, we consider the graph to have asymmetric cost and demand functions for servicing and deadheading. Such asymmetric functions can also model one-way streets for ground robots.

Large graphs using multi-depot formulation: When the network is large, it may not be possible to service the entire network from a single *depot* (or home) location. A standard preprocessing approach is to divide the input graph into small subgraphs using a clustering-based approach, assign a depot location for each cluster, and treat each subgraph as an independent single-depot problem. However, these preprocessing steps require a *similarity* function for clustering, which does not typically consider the nuances of routing problems, such as the minimization of the route cost and the constraints on the resources, and so can lead to inefficient routes. Furthermore, as the individual subgraphs are solved independently, routes that use edges from different subgraphs are not permitted, thereby significantly restricting the solution space. To address large-

scale graphs, this paper presents a multi-depot ILP formulation where the robots have the flexibility to optimize their routes by starting and ending their routes at one of several depots. We further develop an efficient multi-depot MEM heuristic algorithm, MD-MEM, which optimizes the total cost of the routes, respects the resource constraints, and considers all the available depot locations for the routes.

Turning costs and nonholonomic constraints: Sharp turns can be costly for robots as they require the robots to slow down, turn, and then accelerate. *Nonholonomic* robots, such as fixed-wing UAVs and underwater vehicles, cannot make sudden turns. Often a postprocessing step is employed to modify computed routes so that nonholonomic constraints are respected. Such procedures, however, are not guaranteed to comply with the resource capacity constraint. To account for these, the paper directly incorporates turning costs and non-holonomic constraints into the multi-depot MEM algorithm.

While the practical aspects discussed above are sometimes addressed separately in motion planning and routing problems, they have not been addressed satisfactorily for the line coverage problem. Often a graph preprocessing or route postprocessing step is employed to resolve the practical constraints. To the best of our knowledge, this paper is the first to develop a unified approach to address the above three practical aspects of the line coverage problem.

The multi-robot line coverage problem is NP-hard, and there are no known efficient approximation algorithms with theoretical bounds, even for simpler variants of the problem. Thus, we establish the efficacy and efficiency of the MEM heuristic algorithm in simulation on two new datasets[1] consisting of road networks from the 50 most populous cities in the world. Illustrative physical experiments are also presented.

---

[1]The dataset is available at:
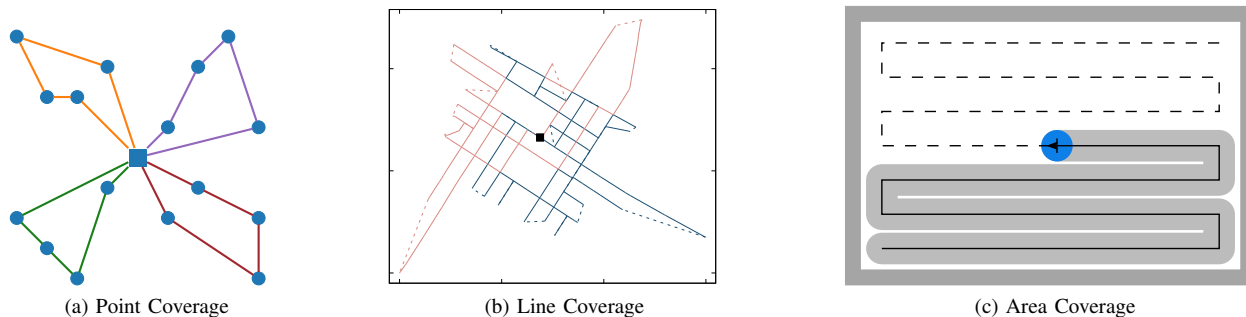https://github.com/UNCCharlotte-CS-Robotics/LineCoverage-dataset

Fig. 2. Three types of features and the corresponding coverage problems: (a) Point coverage is the coverage of zero-dimensional point features and is commonly solved using node routing algorithms. (b) Line coverage is the coverage of one-dimensional line features and belongs to the broad class of arc routing problems. (c) Area coverage is the coverage of two-dimensional regions, often solved using computational geometry techniques.

We provide an open-source implementation[2] of our algorithms.

Building on our earlier publication [3], this paper (1) simplifies and improves the ILP formulation, (2) addresses large graphs with multiple depots, (3) incorporates turning costs and nonholonomic constraints, (4) provides extensive simulation results, and (5) validates the algorithms in experiments for the multi-robot line coverage problem. The conference paper addressed large environments by partitioning the input graph into smaller subgraphs and then solving for each subgraph. However, the approach does not consider routing constraints in the partitioning step and, therefore, has several limitations that lead to inefficient routes due to the restricted solution space. This paper addresses the multi-depot line coverage problem within the routing algorithm so that a preprocessing step for graph partitioning is no longer needed, thereby generating efficient routes. We further incorporate turning costs and nonholonomic constraints into the formulation and the algorithms. This leads to the first algorithm for the line coverage problem that has a unified approach for (1) asymmetry in costs and demands, (2) large graphs with multiple depots, and (3) turning costs and nonholonomic constraints.

The rest of the paper is organized as follows. The related work is discussed in Section II. The multi-robot line coverage problem is formally described in Section III, along with ILP formulation. Section IV develops the constructive heuristic algorithm in stages, resulting in a unified and efficient approach. The simulations and experiments are discussed in Section V. Section VI concludes the paper.

II. RELATED WORK

In a *coverage* application, the robots are required to visit specified features in the environment. Environments may have features of interest that can be represented as points, lines or curves, and areas, resulting in three distinct types of coverage problems, as illustrated in Figure 2. We provide a brief discussion on node routing problems, often used to solve the coverage of points. Next, the arc routing problems (ARPs) and their relation to the line coverage problem are discussed. Finally, the application of line coverage to area coverage problems in robotics is introduced.

[2]The source code is available at:
https://github.com/UNCCharlotte-CS-Robotics/LineCoverage-library

A. Point Coverage and Node Routing Problems

Point coverage is the coverage of point features. Single robot versions of the problem, without resource constraints, can be modeled on a graph and solved as a traveling salesperson problem (TSP) [4]. Since deciding whether a graph is Hamiltonian is NP-complete [5], a reduction from the Hamiltonian cycle problem shows that the TSP is NP-hard, and approximating it to a constant factor is also NP-hard. Christofides [6] gave a 3/2-approximation algorithm for the TSP when the edge costs satisfy the triangle inequality. When the edge costs are asymmetric, the problem is called the asymmetric traveling salesperson problem (ATSP); this is much harder than its symmetric counterpart. Traub and Vygen [7] recently gave a $(22 + \epsilon)$-approximation algorithm for the ATSP under triangle inequality conditions. The line coverage problem can model points by treating each point as a degenerate edge, i.e., a required edge of zero cost, and connecting them to each other by non-required edges. This makes the ATSP a special case of the line coverage problem, and the NP-hardness of the ATSP applies to the line coverage problem.

In the orienteering problem [8], rewards are associated with the nodes, and the goal is to maximize the total reward while respecting a given resource constraint. This problem is also NP-hard and approximation algorithms have been developed [9], [10]. Variations of the orienteering problem, such as with Dubins curves [11], multiple vehicles [12], [13], and correlations on arcs [14] have also been studied. However, the tour is not guaranteed to visit every node due to the resource constraint. Hence, the orienteering problem cannot be used for point coverage, where *all* points must be visited. In contrast, our formulation of the line coverage problem generates routes that together cover all the required edges and potentially required nodes (points) while ensuring each route respects the resource constraints.

There are additional variants of node routing problems, such as the generalized traveling salesperson problem, the vehicle routing problem [15], the gas station problem [16], and routing with fuel depots [17]. However, they do not consider asymmetric costs, multiple depots, resource capacity, and turning costs within a unified framework. Furthermore, they are fundamentally different from line coverage as they do not require the robot to traverse a set of required edges.

B. Arc Routing Problems

The line coverage problem belongs to the broad class of arc routing problems (ARPs). The ARPs are usually applied to transportation problems in which servicing is related to tasks such as delivery and pick-up of goods [1]. In such problems, the costs are associated with the travel distances, and they have the same value for servicing and deadheading. Furthermore, the demands are associated with the load the vehicles can carry, e.g., the capacity of a garbage collection truck. Therefore, the deadheading of an edge does not affect the capacity. In contrast, the capacity in the line coverage problem is associated with battery life, and demands are incurred in both modes of travel.

The Chinese postman problem (CPP), the simplest of the ARPs, is to find an optimal tour such that every edge in an undirected and connected graph is traversed at least once. Edmonds and Johnson [18] used matching and network flow to solve the CPP on undirected, directed, and Eulerian mixed graphs. In the rural postman problem (RPP), the task is to service a subset of edges in a graph. Frederickson [19] gave a 3/2-approximation algorithm based on the algorithm given by Christofides [6] for the TSP. The asymmetric RPP, i.e., RPP on a graph with asymmetric edge costs, is much harder than the symmetric counterpart. The formulation for the single robot line coverage problem, given by Agarwal and Akella [20], [21], can be used to solve asymmetric RPP. They developed algorithms based on the structure of the required graph—the graph induced by the set of required edges. A 2-approximation algorithm for the case of a connected required graph and an $(\alpha(C) + 2)$-approximation algorithm for a required graph with C connected components were developed. Here, $\alpha(C)$ is the approximation factor for an algorithm for the ATSP with triangle inequality conditions on the edge cost. The generalized routing problem (GRP) explicitly considers required nodes and required edges in the graph, and new heuristic algorithms have been proposed recently [22].

Easton and Burdick [23] introduced kRPP, the RPP with k vehicles. They modeled coverage of 2D object boundaries as a kRPP and presented a cluster first and route-second heuristic. The kRPP does not consider the capacity of the robot (vehicle). The capacitated arc routing problem (CARP), introduced by Golden and Wong [24], considers k vehicles of a given capacity Q. The required edges have a demand associated with them, and the total demand of a route should be less than the capacity. The CARP is a special case of the line coverage problem—there are no deadheading demands, and all costs and demands are symmetric. Since the problem is NP-hard [24], several heuristic algorithms have been developed for the CARP [1]. Wøhlk [25] presented a 7/2-approximation algorithm for the problem. Several variants of the ARP partially or individually consider a subset of the characteristics of the line coverage problem. Gouveia et al. [26] presented a lower bound approach for the CARP on mixed graphs using a mixed integer linear program (MILP). The CARP-DD, introduced by Kirlik and Sipahioglu [27], considers deadheading demands for the CARP. The line coverage problem is a generalization of all these ARPs, as shown in Table I.

TABLE I
THE LINE COVERAGE PROBLEM WITH ITS SPECIAL CASES

Literature	Service		Deadheading	
	Cost	Demand	Cost	Demand
kRPP [23]	S	–	=	–
CARP [24]	S	S	=	–
MDCARP [28]	S	S	=	–
CARP-DD [27]	S	S	=	S
Mixed-CARP [26]	A	A	A	–
Line coverage problem	**A**	**A**	**A**	**A**

S: symmetric, A: asymmetric, and –: not considered.
=: The deadheading costs are equal to the service costs.

In the multi-depot capacitated arc routing problem (MD-CARP), a set of depot locations is given as input. The vehicles can start and end their route at any depot, and usually, the requirement is to return to the start depot at the end of the route. The MDCARP is especially relevant in large-scale applications where servicing a network may not be optimal or even feasible from a single location. Given the low battery life of robots, compared to fueled vehicles, the MDCARP becomes relevant even for moderately sized graphs. One common technique is to cluster the input graph into smaller subgraphs and assign a single depot for each subgraph. The algorithms for the CARP are then used on each of the subgraphs independently. Such techniques have also been described as districting by Muyldermans et al. [28] and as sectoring by Mourão et al. [29].

Several exact and metaheuristic algorithms have been proposed for the ARPs. They are covered in the survey paper by Corberán and Prins [30] and the monograph by Corberán and Laporte [1]. Exact approaches include branch-and-bound with cutting planes, branch-and-price, and column generation. The problem is NP-hard, and these exponential-time algorithms are not suitable for large-scale robotics applications. Metaheuristic algorithms, such as scatter search, tabu search, and variable neighborhood descent, have been used for ARPs. Similar to the exact methods, these can require significant computation resources. Moreover, they typically require a good initial solution as an additional input to upper bound the optimal cost. The heuristic algorithms presented in this paper can provide such an initial solution and be used as a fast sub-routine to generate intermediate solutions.

Algorithms based on minimum-cost perfect matching and minimum-cost flow problems are used widely to solve ARPs. They generate a set of Eulerian digraphs, and a robot route can be generated by computing an Eulerian tour for each such digraph. Incorporating multiple depots, turning costs, and nonholonomic constraints is a significant challenge when using these approaches. In contrast, this paper develops a constructive heuristic algorithm, Merge-Embed-Merge (MEM), which maintains a set of routes for the robots and constructively merges pairs of routes to form larger routes. The algorithm is very fast and gives solutions of high quality. Using the basic structure of the algorithm for asymmetric costs, we

solve the multi-depot version of the line coverage problem for large graphs and incorporate turning costs and nonholonomic constraints. To the best of our knowledge, this is the first algorithm to address these challenging practical aspects of line coverage in a unified manner within a single algorithm.

C. Line Coverage in Robotics

There has been little work on using robots for line coverage tasks. Dille and Singh [31] presented algorithms to perform coverage of a road network using a single aerial robot with kinematic constraints. They modeled the problem through the tessellation of the road segments by circles of radius corresponding to the sensor footprint and computing a subset of the circular regions covering the entire road network. Algorithms for node routing problems, such as the TSP and the multiple TSP, are then used to find the routes for the robots. Oh et al. [32] proposed an MILP formulation and a heuristic algorithm for the coverage of road networks. The nearest insertion heuristic, originally designed for TSP, finds a sequence of edges to be visited while incorporating Dubins curves for nonholonomic robots. The sequence is split across a team of robots using an auction algorithm. Algorithms for kRPP were developed for boundary inspection with multiple robots by Easton and Burdick [23]. Williams and Burdick [33] developed algorithms for boundary inspection while considering revisions to the path plan for the robots to account for environmental changes. Xu and Stentz [34] use CPP and RPP formulations for environmental coverage and consider the case of incomplete prior map information. They extended this work to multiple robots [35] using k-means clustering to decompose the environment into smaller components, similar to the approach in [23]. Campbell et al. [36] presented an application of ARPs to cover road networks using a single UAV. They discretize the required edges to allow the UAV to service an edge in parts.

The above work illustrates line coverage applications in robotics. However, line coverage has not been studied as extensively as the area coverage problem or node routing problems. Current work does not usually consider the battery life of the robots and thus may not be suitable for large-scale applications. In contrast, we model battery life as capacity (resource) constraints and consider multiple depots, enabling solutions for large networks. Furthermore, we allow demands on resources while deadheading, and asymmetric functions for costs and demands.

D. Arc Routing Problems in Area Coverage

ARPs have been used in area coverage problems as a subroutine to generate efficient robot routes. Arkin et al. [37] used the CPP to find a route for the milling problem, a variant of the area coverage problem wherein the tool is constrained within the workspace. Mannadiar and Rekleitis [38] performed a cell decomposition and computed a Reeb graph. The edges of the graph correspond to the cells in the decomposition. The problem of visiting the cells was then formulated as the CPP. Karapetyan et al. [39] used the CPP to compute a large Euler tour and then decomposed the tour into smaller ones using

an algorithm given by Frederickson et al. [40]. Recently, we presented a new approach [41] for the area coverage problem by generating service tracks in the environment after performing cell decomposition. These service tracks were modeled as required edges for the line coverage problem, and the MEM algorithm was used to generate efficient routes for multiple resource-constrained robots. Using the above technique, the algorithms presented in this paper for multiple depots and nonholonomic robots apply to the area coverage problem.

III. THE LINE COVERAGE PROBLEM

We pose the line coverage problem as an optimization problem on a graph. The environment comprises linear features (line segments or curves) that need to be serviced by a homogeneous team of robots. It is modeled as an undirected and connected graph $G = (V, E, E_r)$, where $E_r \subseteq E$ is the set of *required edges* representing the linear features. The graph may have edges that do not require servicing, and the robots can use them to optimize their routes; these are the *non-required edges* given by $E_n = E \setminus E_r$. The set E can contain parallel edges, i.e., we allow for G to be a multigraph. The set of vertices V consists of edge endpoints, edge intersections, and depot locations. The *depots* $V_d \subseteq V$ are a subset of vertices at which the robots start and end their routes.

For each edge e in E we associate two directional arcs a_e and $\bar{a}_e$ that are opposite in direction to one another. There are two modes of travel for a robot—servicing and deadheading. A robot is said to be *servicing* an edge when it performs task-specific actions such as taking images along the edge. We associate two binary variables $s_{a_e}^k$ and $s_{\bar{a}_e}^k$ with servicing an edge $e \in E_r$ by a route k: if a robot, executing the route k, services edge e in the direction a_e, then $s_{a_e}^k$ is 1 and 0 otherwise; similarly for the direction $\bar{a}_e$. Let K be an upper bound on the total number of routes. Since each required edge is required to be serviced exactly once, we have:

$$\sum_{k=1}^{K} \left(s_{a_e}^k + s_{\bar{a}_e}^k \right) = 1, \quad \forall e \in E_r \tag{1}$$

If a robot traverses an edge without servicing it, the robot is said to be *deadheading*, e.g., this occurs when a robot travels from its depot to an edge to be serviced. We associate two non-negative integer variables $d_{a_e}^k$ and $d_{\bar{a}_e}^k$ with deadheading an edge $e \in E$ in route k. The edges can be deadheaded any number of times. The task is to compute a set of K routes $\Pi = \{\pi^1, \ldots, \pi^K\}$. If we have K robots, then each route is executed by a distinct robot. If not, some robots may have to execute multiple routes to cover all the required edges. The formulation does not restrict the number of robots.

Servicing an edge $e \in E_r$ in the direction a_e incurs a service cost $c_s(a_e)$; similarly for the direction $\bar{a}_e$. Analogously, deadheading an edge $e \in E$ incurs deadhead costs $c_d(a_e)$ and $c_d(\bar{a}_e)$. A robot may need to travel slower while performing task-specific actions, resulting in higher costs for servicing than deadheading. Thus, the two cost functions can differ. The costs are associated with a minimization objective function of the optimization problem, such as total travel time.

The cost of a route $\pi^k \in \Pi$ is given as:

$$
c(\pi^k) = \sum_{e \in E_r} \left[c_s(a_e)\, s_{a_e}^k + c_s(\bar{a}_e)\, s_{\bar{a}_e}^k \right]
$$
$$
+ \sum_{e \in E} \left[c_d(a_e)\, d_{a_e}^k + c_d(\bar{a}_e)\, d_{\bar{a}_e}^k \right] \tag{2}
$$

Each robot is constrained by a resource such as operation time, total travel distance, or battery life. Such a constraint is represented by a *budget* or *capacity* Q for each robot. The consumption of resources is modeled by demand functions $q_s(a_e)$ for servicing and $q_d(a_e)$ for deadheading an edge $e \in E$ in the direction a_e. The total demand incurred by a robot for a route π^k must be less than the capacity:

$$
q(\pi^k) = \sum_{e \in E_r} \left[q_s(a_e)\, s_{a_e}^k + q_s(\bar{a}_e)\, s_{\bar{a}_e}^k \right]
$$
$$
+ \sum_{e \in E} \left[q_d(a_e)\, d_{a_e}^k + q_d(\bar{a}_e)\, d_{\bar{a}_e}^k \right] \leq Q \tag{3}
$$

We consider the edge costs and demands for both servicing and deadheading to be direction-dependent, i.e., the graph is asymmetric. For example, $c_s(a_e)$ can differ from $c_s(\bar{a}_e)$. Asymmetry in graphs can occur when modeling wind for aerial robots or terrain for ground robots. It also allows us to model one-way streets, directed graphs, and mixed graphs in general. This can be achieved by setting the cost and demand in the prohibited direction to be a large constant. The traversal of edges can be modeled using any function, such as constant velocity or cubic trajectories, and travel time can be used as the cost function. Similarly, demands and capacity can be specified in terms of battery life. Such functions can also incorporate wind and terrain information [42], [43]. In general, the costs and demands are allowed to be arbitrary non-negative constants. If we also have point features V_f in the environment, we add an artificial edge (v, v) for each point feature $v \in V_f$. The cost and the resource demand for servicing such an artificial edge would be the same as that of servicing the point feature, and the cost and demand for deadheading is set to zero. This transformation allows modeling both the point and the line features in the same formulation.

Definition: Given an undirected and connected graph $G = (V, E, E_r)$, the *line coverage problem* is to find a set of coverage routes Π that services each required edge in E_r exactly once and minimizes the total cost of the routes, while respecting the resource constraints.

A. Integer Linear Programming Formulation

An integer linear program (ILP) is a formulation for optimization problems with integer variables, a linear objective function, and a set of linear constraints. An ILP formulation provides a concise mathematical description of the problem, and solving the ILP gives an optimal solution to an instance of the problem if the instance has feasible solutions.

1) Variables: We have the following variables for the ILP.

- Binary service variables $s_{a_e}^k, s_{\bar{a}_e}^k \in \{0, 1\}$ for each required edge $e \in E_r$ and each route k.
- Integer deadheading variables $d_{a_e}^k, d_{\bar{a}_e}^k \in \mathbb{N} \cup \{0\}$ for each edge $e \in E$ and each route k.

- Integer flow variables $z_{a_e}^k, z_{\bar{a}_e}^k \in \mathbb{N} \cup \{0\}$ for each edge $e \in E$ and each route k. The flow variables are used in connectivity constraints to ensure that routes are connected to the depots.

For now, we assume that all the robots start and end their routes at the same depot location $v_0 \in V$. We will generalize the formulation to multiple depots in the following subsection.

2) Routing Constraints: The routing constraints ensure connectivity of a route to the depot and eliminate sub-tours. We represent the assigned depot for each route k by $v_d^k \in V_d \subseteq V$. For the single depot problem, considered in this subsection, $v_d^k = v_0$ for all k.

For ease of notation, we define the following sets:

$$
\mathcal{A} = \bigcup_{e \in E} \{a_e, \bar{a}_e\}, \quad \mathcal{A}_r = \bigcup_{e \in E_r} \{a_e, \bar{a}_e\},
$$
$$
H(\mathcal{A}, v) = \text{arcs in } \mathcal{A} \text{ with } v \text{ as the head, and}
$$
$$
T(\mathcal{A}, v) = \text{arcs in } \mathcal{A} \text{ with } v \text{ as the tail.}
$$

Here, $\mathcal{A}$ is the set of all arcs, and $\mathcal{A}_r$ is the set of arcs corresponding to required edges.

We have the following set of routing constraints for each route $k \in \{1, \ldots, K\}$:

$$
\sum_{a \in T(\mathcal{A}, v_d^k)} z_a^k = \sum_{a \in \mathcal{A}_r} s_a^k \tag{4}
$$

$$
\sum_{a \in H(\mathcal{A}, v)} z_a^k - \sum_{a \in T(\mathcal{A}, v)} z_a^k = \sum_{a \in H(\mathcal{A}_r, v)} s_a^k, \quad \forall v \in V \setminus \{v_d^k\} \tag{5}
$$

$$
z_a^k \leq \sum_{a \in \mathcal{A}_r} s_a^k, \quad \forall a \in \mathcal{A} \tag{6}
$$

$$
z_a^k \leq |E| d_a^k, \quad \forall a \in \mathcal{A} \setminus \mathcal{A}_r \tag{7}
$$

$$
z_a^k \leq |E|(s_a^k + d_a^k), \quad \forall a \in \mathcal{A}_r \tag{8}
$$

$$
\sum_{a \in H(\mathcal{A}_r, v)} s_a^k + \sum_{a \in H(\mathcal{A}, v)} d_a^k = \sum_{a \in T(\mathcal{A}_r, v)} s_a^k + \sum_{a \in T(\mathcal{A}, v)} d_a^k, \quad \forall v \in V \tag{9}
$$

The constraints (4) to (8) are generalized flow constraints that together ensure the connectivity of the route to the depot and prohibit any sub-tours. The variables z_a^k are flow variables for each edge direction. Constraint (4) defines the amount of flow being released from the depot vertex $v_d^k = v_0$, which acts as a source of the flow. For any vertex v (other than the depot vertex), a flow equal to the number of servicing arcs, with v as the head, is absorbed by the vertex. This is expressed in constraints (5). The amount of flow through an arc is limited by constraints (6) to (8). An edge has a positive flow if and only if it is traversed. Finally, the vertex symmetry constraints (9) ensure that the number of arcs entering a vertex is the same as the number of arcs leaving it.

ILP Formulation: The objective function of the line coverage problem is the total cost of the routes, with K as an upper bound on the total number of routes. We can now pose the line coverage problem as an optimization problem formulated as an ILP:

Minimize:

$$
\sum_{k=1}^{K} c(\pi^k) = \sum_{k=1}^{K} \left[\sum_{a \in \mathcal{A}_r} c_s(a) s_a^k + \sum_{a \in \mathcal{A}} c_d(a) d_a^k \right] \tag{10}
$$

subject to:

$$\sum_{k=1}^{K} \left(s_{a_e}^k + s_{\bar{a}_e}^k \right) = 1, \quad \forall e \in E_r \tag{11}$$

$$q(\pi^k) = \sum_{a \in \mathcal{A}_r} q_s(a) \, s_a^k + \sum_{a \in \mathcal{A}} q_d(a) \, d_a^k \leq Q, \quad \forall k \tag{12}$$

Routing constraints (4) to (9) for each robot k (13)

$$s_a^k \in \{0, 1\}, \quad \forall a \in \mathcal{A}_r \quad \forall k \tag{14}$$

$$d_a^k, z_a^k \in \mathbb{N} \cup \{0\}, \quad \forall a \in \mathcal{A}, \quad \forall k \tag{15}$$

B. Multi-Depot Formulation

In graphs spanning over a large area, it may be inefficient or even infeasible to service all the locations from a single location. Hence, we develop a multi-depot formulation, wherein we are given a set of potential depot locations $V_d \subseteq V$, and a route can be assigned to any of the depots to start and end the route. The problem can be formulated by adding assignment constraints—each route needs to be assigned exactly one depot location from V_d.

In general, the entire vertex set V could be the set of potential depot locations. However, this would increase the complexity of the problem significantly. Instead, a smaller subset of vertices, ideally of size K or less, can help in reducing the complexity while also providing high-quality solutions. These locations can be selected from the field of operation based on terrain data or by clustering the vertices or the edges.

We introduce the binary variable x_d^k, which is 1 when the depot $v_d \in V_d$ is assigned to route k, and 0 otherwise.

$$\sum_{v_d \in V_d} x_d^k = 1, \quad \forall k \tag{16}$$

$$v_d^k = \sum_{v_d \in V_d} v_d \, x_d^k, \quad \forall k \tag{17}$$

$$x_d^k \in \{0, 1\}, \quad \forall v_d \in V_d, \forall k \tag{18}$$

Constraints (16) ensure that exactly one depot is assigned to each route. Contraints (17) assign the depots to the routes—v_d^k is set to the depot v_d for which x_d^k is 1.

Routing constraints (4) and (5) depend on the assigned depot v_d^k. The constraint (4) is active only for the assigned depot, whereas constraints (5) are active for all the vertices except the assigned depot. These can be resolved by multiplying the variable x_d^k to both sides of the constraint (4), and the expression $(1 - x_d^k)$ to both sides of constraints (5).

However, this will result in a quadratic set of constraints, and the problem will become non-linear. Although such quadratic constraints can be converted to linear constraints by introducing additional binary variables and large constants, it would significantly increase the size of the problem. As the motivation for the multi-depot problem is to solve large instances, it is not beneficial to formulate the problem as an ILP, which becomes harder to solve for instances with a large number of variables and constraints. The difficulty of solving such variants of the problem further motivates the development of versatile constructive heuristic algorithms that can efficiently solve the problem for large instances.

IV. CONSTRUCTIVE ALGORITHMS FOR LINE COVERAGE

The line coverage problem and its variants are NP-hard problems. Computing optimal solutions, e.g., using an ILP formulation, is usually feasible only for small instances. This motivates us to develop heuristic algorithms to compute high-quality solutions for large instances. Moreover, the algorithm is constructive—it maintains a set of feasible routes and iteratively merges a pair of routes to form a new larger route. The overall algorithm for the line coverage problem with multiple robots is built in stages. We first present in detail the Merge-Embed-Merge (MEM) algorithm for graphs with asymmetric costs and demands. Using the constructive nature of the MEM algorithm, we then develop the multi-depot MEM (MD-MEM) algorithm for large graphs. Finally, we incorporate turning costs and nonholonomic constraints to develop a unified MD-MEM-Turns algorithm.

A. Merge-Embed-Merge: A Constructive Heuristic

This section develops a new algorithm, Merge-Embed-Merge (MEM), for the line coverage problem. The underlying concept is to maintain a set of feasible routes; initially, a route is created for each required edge. Subsequently, routes are merged together greedily to form a smaller set of routes. This concept of merging was first presented by Clarke and Wright [44] for the capacitated vehicle routing problem, which was later adopted in the *Augment-Merge* heuristic by Golden et al. [45] for the CARP. However, the Augment-Merge algorithm cannot handle the asymmetric costs and demands and the deadheading demands of the line coverage problem. Furthermore, the heuristic degrades rapidly with instance size, especially when the set of required edges is much smaller than the entire edge set [1]. We improve this by including an *embed* step in our algorithm.

The MEM algorithm, given in Algorithm 1, comprises four components: (1) *initialization* of routes, (2) computation of *savings*, (3) *merging* two routes to form a new route, and (4) *embedding* the newly merged route. A max-heap data structure is used to greedily decide the two routes to be merged and embed new routes.

We begin by discussing the *representation of routes* in the algorithm. A route is represented by a sequence of required arcs that are to be serviced by the robot traversing the route. As the costs and the demands can be direction-dependent, arcs are used instead of the corresponding required edges. The robot starts at the depot, travels to the starting vertex of the first required arc, services the sequence of required arcs, and returns to the depot. Consider a route R_p, as shown in Figure 3: the vertex i is the starting vertex of the first required arc a_s, and the vertex j is the end vertex of the last required arc a_l in the sequence given by R_p. The robot will deadhead from the depot v_0 to vertex i, service the required arcs starting from i up to the last required arc ending at vertex j, and then deadhead back to the depot v_0. Note that the paths $v_0 \rightarrow i$ and $j \rightarrow v_0$ are the shortest paths and may contain several arcs, all deadheaded. If two successive required arcs a_t and a_{t+1} are not adjacent, the robot deadheads along the shortest path from the end vertex of a_t to the start vertex of a_{t+1}.

This article has been accepted for publication in IEEE Transactions on Robotics. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TRO.2024.3355802

8

Thus, the sequence $i \to j$ may involve deadheading between the constituent required arcs. The cost of the route R_p is:

$$c(R_p) = c_d(v_0, i) + c_s(R_p) + c_d(j, v_0) + \lambda \qquad (19)$$

The shortest path gives the costs of deadheading from and to the depot. The cost of servicing the sequence of required arcs in R_p is given by $c_s(R_p)$, which may include deadheadings between non-adjacent required arcs. An additional constant cost λ representing route setup cost is added to the route cost. The route setup cost represents the additional time required to launch a robot from a depot and helps in reducing the number of routes during the merging process.
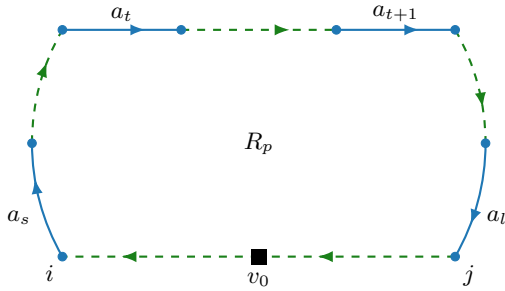

Fig. 3. Representation of a route R_p as a sequence of arcs corresponding to required edges: The required arcs are shown as solid blue lines, and deadheadings are shown as dashed green lines. The route internally includes deadheadings given by shortest paths to and from the depot v_0 (black square). The route may have deadheadings between two non-adjacent required arcs.

1) Initialization of Routes: The MEM algorithm (Algorithm 1) constructs a route for each required edge in the initialization step (lines 2–7). Each edge $e \in E_r$ has two arcs a_e and $\bar{a}_e$ associated with it, representing the two travel directions. Let i and j denote the tail and the head of the arc a_e, respectively (line 3). Then the route for servicing a_e comprises the shortest path from the depot v_0 to the tail vertex i, servicing of arc a_e, and the shortest path from the head vertex j to the depot. In the other direction, the route comprises the shortest path from the depot to vertex j, servicing of arc $\bar{a}_e$, and the shortest path from vertex i to the depot. Since the costs are asymmetric, of the two routes, the one with the lower cost is selected (line 6). It is assumed that the demand for the initial routes is less than the capacity; otherwise, the instance does not have a feasible solution. A constant route setup cost λ is added to the route cost. The routes are stored in the list $\mathcal{R}$, and there are $m = |E_r|$ routes initially. The particular case where an edge can be serviced in only one direction can be handled appropriately in the initialization step.

2) Computation of Savings: Consider two routes R_p and R_q, with tail and head vertices given by t_p, h_p and t_q, h_q, as potential candidates for merging. There are eight possible permutations to merge the two routes, of which four are shown in Figure 4. The remaining four ways consist of routes in reverse directions. The first merged route R_{pq} and its cost

---

Algorithm 1: Merge-Embed-Merge (MEM)

Input : $G = (V, E, E_r)$, costs, demands, capacity Q
Output : Coverage routes $\mathcal{R}$, where each tour $R \in \mathcal{R}$ is a sequence of required edges

```
/* Initialization of routes                    */
1  𝓡 ← ∅; k ← 1;
2  for e ∈ E_r do           // a_e and ā_e are arcs for e
3      i ← tail(a_e); j ← head(a_e);
4      c ← c_d(v_0, i) + c_s(a_e) + c_d(j, v_0) + λ;
5      c̄ ← c_d(v_0, j) + c_s(ā_e) + c_d(i, v_0) + λ;
6      if c ≤ c̄ then R_k ← a_e else R_k ← ā_e;
7      𝓡.push(R_k); k ← k + 1;
   /* Compute savings                           */
8  S ← ∅;
9  foreach pair of tours R_p, R_q do
10     Compute saving s_pq for R_p ⊎ R_q;
11     if s_pq ≥ 0 and demand(R_p ⊎ R_q) ≤ Q then
12         S.push((p, q, s_pq));
13 make_heap(S);              // max-heap
   /* Repeated Merge and Embed                  */
14 while S ≠ ∅ do
15     (p, q, s) ← S.extract-max( );
16     if R_p ≠ ∅ and R_q ≠ ∅ then
           /* Merge                             */
17         R_k ← R_p ⊎ R_q;
18         𝓡.push(R_k); k ← k + 1;
19         R_p ← ∅; R_q ← ∅;
           /* Embed                             */
20         foreach tour R_i with i ≠ k and R_i ≠ ∅ do
21             Compute saving s_ki for R_k ⊎ R_i;
22             if s_ki ≥ 0 and demand(R_k ⊎ R_i) ≤ Q then
23                 S.insert((k, i, s_ki));
24 Remove empty routes from 𝓡;
```

$\text{cost}(R_{pq})$ are:

$$R_{pq} := R_p \uplus R_q := v_0 \to t_p \xrightarrow{R_p} h_p \to t_q \xrightarrow{R_q} h_q \to v_0$$
$$\text{cost}(R_{pq}) = c_d(v_0, t_p) + c_s(R_p) + c_d(h_p, t_q)$$
$$+ c_s(R_q) + c_d(h_q, v_0) + \lambda$$

Such a merge can have potential *saving* in cost since we no longer require $h_p \to v_0$ and $v_0 \to t_q$. There is also a cost-saving of a route setup cost λ as we have a single route instead of two. Thus, the net saving in cost s_{pq} for merging routes R_p and R_q is given by $\text{cost}(R_p) + \text{cost}(R_q) - \text{cost}(R_{pq})$. However, the cost savings are affected by the direction of the edges due to asymmetry in the costs. Hence, we need to consider all eight permutations for merging two routes. Some of these permutations might not satisfy the resource capacity constraint, i.e., the total demand of the merged route may be more than the capacity Q. The merged route that satisfies the resource capacity constraint and has the maximum cost savings is denoted by $R_p \uplus R_q$. If no such combination exists, then $R_p \uplus R_q = \emptyset$ and savings $s_{pq} = -\infty$. Note that the algorithm allows reversal of the service directions, i.e., the service arcs of a constituent route R_p can be reversed in the merged route R_{pq}. Hence, the direction of the arcs in the initialization step is not fixed, and the algorithm chooses the direction that yields the maximum cost savings in each iteration.
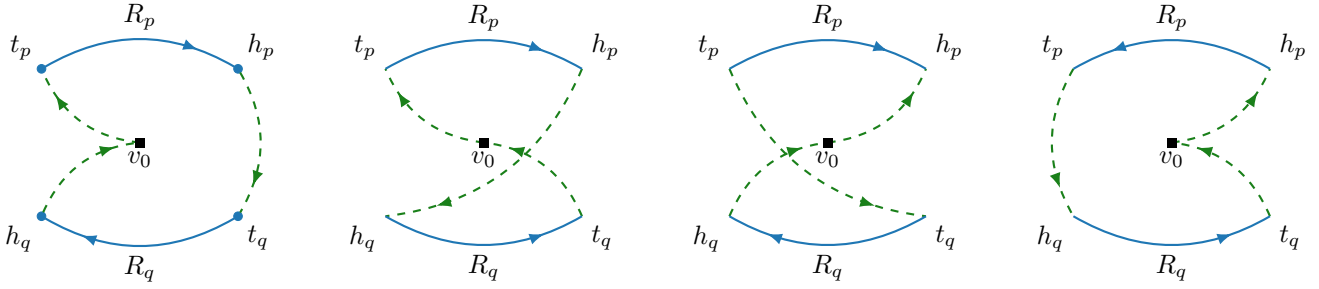
Fig. 4. The figure shows four of the eight permutations to merge two routes R_p and R_q. The remaining four permutations consist of the shown permutations in the reverse directions. The tail and the head vertices for R_p are t_p and h_p, respectively. Similarly, t_q and h_q are defined for R_q. The first merged route is $v_0 \to t_p \xrightarrow{R_p} h_p \to t_q \xrightarrow{R_q} h_q \to v_0$ and its reverse direction route is $v_0 \to h_q \xrightarrow{\overline{R_q}} t_q \to h_p \xrightarrow{\overline{R_p}} t_p \to v_0$. The savings for merging two routes come from potentially reduced deadheading to and from the depot.

For each pair of routes that yield a feasible merged route, the maximum saving in cost is computed and stored as a tuple (p, q, s_{pq}), where p and q correspond to the routes considered, and s_{pq} is the corresponding saving (lines 9–12). These $m(m-1)/2$ tuples are stored in a binary max-heap data structure S, which can be built in $\mathcal{O}(m^2)$ computation time (line 13).

Next, the merge and embed steps are executed repeatedly until no further merges are possible (lines 14–23).

3) Merge: The pair of routes with maximum cost savings is selected to form a new merged route, thereby maximizing immediate savings. The maximum element from the max-heap S is extracted (line 15), and the constituent routes are merged if both are non-empty (lines 16–17). The merged route R_k is added to the list of routes $\mathcal{R}$ (line 18). The constituent routes are set to $\emptyset$ so that they are no longer considered for future merges (line 19). Setting a route to $\emptyset$ does not modify the max-heap data structure, but only the corresponding element in the list of routes $\mathcal{R}$. Empty routes are filtered when the elements from the max-heap are extracted (line 16). The complexity of the merge step is $\mathcal{O}(\log|S|)$, where $|S|$ is the number of elements in S.

4) Embed: We consider merging existing non-empty routes with the newly merged route R_k in the embed step (lines 20–23). Potential cost savings are computed for the new route R_k if merged with the other non-empty routes in the list $\mathcal{R}$. New tuples (k, i, s_{ki}) are generated and inserted into the max-heap S, if merging satisfies the capacity constraint and the cost saving is non-negative (lines 22–23). As there are $|\mathcal{R}|-1$ such new tuples, the embed step has a computational complexity of $\mathcal{O}\left(|\mathcal{R}| \log\left(|S| + |\mathcal{R}|\right)\right)$.

The merge and embed components are executed until no further merges are possible, i.e., $S = \emptyset$. The maximum number of routes in the list $\mathcal{R}$ is upper bounded by $2m$, with at most m non-empty routes at any iteration. Here, $m = |E_r|$ is the number of required edges. The maximum number of elements in the max-heap S is $\mathcal{O}(m^2)$. Thus the complexity of the repeated merge-embed component over all possible merges is $\mathcal{O}(m^2 \log m)$, which is also the overall complexity of the algorithm. Depending on the instance, one may need to compute the shortest deadheading paths between all pairs of vertices. This can be done using the Floyd-Warshall algorithm in $\mathcal{O}(|V|^3)$ computation time [46]. The MEM algorithm does not have an approximation guarantee; however, we show in Section V that the algorithm performs well in practice.

There are two essential characteristics of the algorithm that have practical benefits: (1) The algorithm maintains a feasible set of routes. These routes can be extracted at any point in the algorithm, giving it the *anytime* property. (2) The number of routes generated by the algorithm does not depend on the number of robots, i.e., the algorithm is agnostic to the number of robots available. It generates the number of routes required to cover the entire environment completely. Thus if a small fleet of robots is available, some of the robots can execute multiple routes by replacing or recharging batteries.

B. Multi-Depot Line Coverage for Large Graphs: MD-MEM

In the above MEM algorithm, we considered a single depot location where all the robots start and end their routes. However, it may not be efficient or feasible to service the required edges representing the linear features from a single location for environments spanning over a large area. Thus, we extend the MEM algorithm to enable multiple depots for the line coverage problem.

A common approach for solving multi-depot problems is to cluster the required edges and create subgraphs [28], [29]; this is, in fact, what we did in our earlier work [3]. Each subgraph is then treated as an instance of the single-depot problem. However, as the generated clusters are treated independently, sometimes multiple routes are generated for a cluster with one of the routes serving only a small number of edges, i.e., the robots are not using their capacity to their full potential. The algorithm does not have the freedom to exchange or transfer some of these edges to nearby clusters. This often leads to inefficient solutions. This limitation is resolved in the multi-depot version of the MEM algorithm by directly considering the depots in the routing process.

We are given as input a set of depot locations $V_d \subseteq V$. These locations can be specified based on operator ease and field constraints, e.g., an operator may prefer to launch aerial robots from high vantage points. Alternatively, the locations can be determined using a k-medoids clustering algorithm. As clustering algorithms are usually non-deterministic, one may choose to perform clustering multiple times to obtain a more

---

Procedure MD-MEM::Initialize

Input : $G = (V, E, E_r)$, depots V_d, costs, demands, capacity Q
Output : Initialized coverage routes $\mathcal{R}$ with assigned depots

1   $\mathcal{R} \leftarrow \emptyset$; $k \leftarrow 1$;
2   **for** $e \in E_r$ **do**   // a_e and $\bar{a}_e$ are arcs for e
3     $R_k \leftarrow \emptyset$; $R_k.\text{cost} \leftarrow \infty$;
4   **for** $v \in V_d$ **do**   // Iterate over depots
5       $i \leftarrow \text{tail}(a_e)$; $j \leftarrow \text{head}(a_e)$;
6       $c \leftarrow c_d(v, i) + c_s(a_e) + c_d(j, v) + \lambda$;
7       $d \leftarrow q_d(v, i) + q_s(a_e) + q_d(j, v)$;
8   **if** $d \leq Q$ **and** $c < R_k.\text{cost}$ **then**
9         $R_k \leftarrow a_e$; $R_k.v_0 \leftarrow v$;   // Route with v_0
10      $\bar{c} \leftarrow c_d(v, j) + c_s(\bar{a}_e) + c_d(i, v) + \lambda$;
11      $\bar{d} \leftarrow q_d(v, j) + q_s(\bar{a}_e) + q_d(i, v)$;
12   **if** $\bar{d} \leq Q$ **and** $\bar{c} < R_k.\text{cost}$ **then**
13        $R_k \leftarrow \bar{a}_e$; $R_k.v_0 \leftarrow v$;   // Route with v_0
14    $\mathcal{R}.\text{push}(R_k)$; $k \leftarrow k + 1$;

Procedure MD-MEM::ComputeSavings

Input : Routes $\mathcal{R}_p$ and $\mathcal{R}_q$, $G = (V, E, E_r)$, depots V_d, capacity Q, costs, demands
Output : Savings s_{pq}

1   $s_{pq} \leftarrow -\infty$;
2   **foreach** $v_0 \in V_d$ **do**
3   Compute saving s_{v_0} for $R_p \uplus R_q$ and depot v_0;
4   **if** $s_{v_0} > s_{pq}$ **and** $\text{demand}(R_p \uplus R_q) \leq Q$ **then**
5       $s_{pq} \leftarrow s_{v_0}$;

---

desirable set of depot locations. Note that while the clustering may be used for assigning depot locations, it is not used to cluster the required edges.

Two modifications need to be made to the MEM algorithm given in Algorithm 1 to integrate the depot selection for the multi-depot line coverage problem: (1) The initialization of the routes, and (2) The computation of savings for merging two routes.

1) Initialization of routes: The initialization process for the multi-depot line coverage problem is given in the procedure MD-MEM::Initialize. For each required edge, we iterate over all the depot locations and compute the cost of servicing the edge in the two directions. The number of such computations is $2|V_d|$. The one with the lowest cost is selected, provided the demand is less than the capacity. It is assumed that a required edge can be serviced from at least one of the depots in at least one direction. The complexity of the initialization step changes from $\mathcal{O}(|E_r|)$ to $\mathcal{O}(|V_d||E_r|)$.

2) Integrated Multiple Depot Route Computation: In the single depot version of the MEM algorithm, we considered eight permutations for merging two routes, four of which are shown in Figure 4. The cost saving for merging two routes R_p and R_q is given by $\text{cost}(R_p) + \text{cost}(R_q) - \text{cost}(R_{pq})$, where R_{pq} is the merged route. In the multi-depot formulation, the cost and the demand of the merged route depend on the depot v_0—the merged route can be assigned to any of the depots, independent of the constituent routes. However, the choice of the depot affects the savings in cost that can be achieved by merging two routes. Thus, we iterate over all

the depots for the multi-depot line coverage problem and check all eight merging permutations for each depot. This gives us $8|V_d|$ computations, and the one with the maximum saving is selected provided it satisfies the capacity constraint. The pseudocode for the computation of savings is given in Procedure MD-MEM::ComputeSavings.

The most expensive component of the MEM algorithm is the embed step. This step involves the computation of savings of a newly merged route with the others in the current set of routes $\mathcal{R}$ (lines 20–23 in Algorithm 1). Thus, the overall complexity of the algorithm changes to $\mathcal{O}(|V_d| m^2 \log m)$. As the computational cost depends on the number of depot locations, keeping their number as small as needed is advantageous. In practice, making the number of depots equal to the number of robots available gives good results.

C. Line Coverage with Turning Costs and Nonholonomic Constraints: MD-MEM-Turns

Until now, we have considered the costs and the demands for traversing the edges to be arbitrary non-negative constants. However, they do not model turning costs and nonholonomic constraints. A smooth trajectory is desired in several robotics applications. Even for differential drive robots, which can perform turns in place, a sharp turn is undesirable as the robot will need to slow down, take a turn, and then accelerate. Smooth paths are often computed as a post-processing step after path planning [47], [48]. However, such post-processing steps are performed individually for each route and do not account for overall cost minimization. Furthermore, the resulting route can violate the constraints on the resource capacity of the robots. Such path-smoothing techniques can be integrated with the MEM algorithm to minimize the total cost and ensure that the demand of a route is within the capacity. This would require modification of the initialization procedure and the computation of savings, similar to the previous section for the multi-depot line coverage problem. However, path smoothing can be an expensive process, and doing so $8|V_d|$ times for the embed step in each iteration may not be practical for applications that require rapid solutions. This section instead illustrates constant-time procedures to generate routes with smooth turns and paths that respect nonholonomic constraints, and integrates them with the MEM algorithm. Furthermore, the costs and the demands for taking turns are incorporated into the MEM algorithm.

1) Smooth Turns and Turning Costs: We first consider the generation of routes with smooth turns for holonomic robots. We will later see that such smooth turns are useful for nonholonomic robots as well. For a specified linear velocity when a robot is in motion, the minimum turning radius for a robot is given by the ratio of its linear and angular velocities; if the turning radius is small, the robot can take sharper turns. When two adjacent edges do not have a sharp corner, the robot can take a smooth turn without deviating too much from the edges, as shown by the green arc in Figure 5(a). The arc is tangential to the two edges. A user-defined parameter $\delta_{\max}$ determines the maximum allowable deviation from the edges, as shown by the red dashed circle. The parameter can be
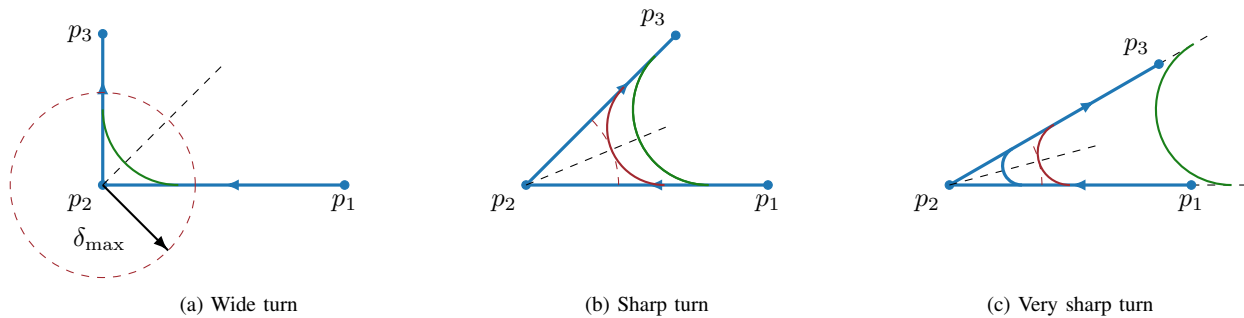
Fig. 5. Smooth turns for a robot traversing two adjacent edges from p_1 to p_3 through p_2. The minimum turning radius is given by the ratio of its linear and angular velocities, and the corresponding arc is shown in green. (a) When the corner is wide, the robot can turn smoothly without changing its velocity and deviating too much from the original path. (b) However, if the turn is sharp, the robot needs to slow down so that the deviation is within the permitted limit $\delta_{\max}$. The optimal turning arc is shown in red. (c) If the turn is very sharp, the robot may not have enough distance to decelerate to the required velocity for the red arc. In the worst case of a 180-degree turn, it may have to come to a complete stop. The innermost blue arc shows the optimal turning arc based on the maximum deceleration and the turning angle.

set based on the sensor field-of-view to ensure coverage of every point on the corresponding linear features. However, if the turn is sharp, the robot needs to slow down so that the maximum deviation is within $\delta_{\max}$, as shown by the red arc in Figure 5(b). This requires the robot to decelerate to achieve the required turning radius by decreasing the linear velocity. When a turn is very sharp, as shown in Figure 5(c), the robot may not have enough length available to decelerate to the required velocity for the red arc corresponding to $\delta_{\max}$. This requires solving a quadratic equation to determine a time-optimal turning arc, which corresponds to the innermost blue arc in the figure. In the worst case of a 180-degree turn, the robot may need to come to a complete stop.

The primary computation step in the MEM algorithm is the computation of savings. Consider two routes R_p and R_q with end edges p and q, respectively, for a particular merge permutation. The algorithm computes the turning cost to go from edge p to edge q. However, such a turn may be sharp and require the robot to decelerate. This, in turn, can affect the velocities on edges $p-1$ and $q-1$ in their respective routes, potentially resulting in a cascading effect that may require computations proportional to the number of edges. To avoid this, we enforce that the robot can decelerate only after it has reached the middle of the first edge p and can accelerate until the middle of the next edge q. By avoiding a cascading effect, this modularizes the computation of the turning costs and ensures constant time computation of savings. As the accelerations of the robots are usually high, and the lengths of the edges are comparatively large in practical applications, we can assume that the robot can come to a complete stop from its full speed within half the length of the edge; otherwise, the robot may not be able to take 180-degree turns.

2) Nonholonomic Robots: Several commonly used robots, such as car-like robots, fixed-wing UAVs, and underwater robots, have nonholonomic constraints and cannot take turns in place. We use a unicycle model of the robots [49] and incorporate Dubins curves for the motion of the robots. Dubins curves are often used to determine optimal paths from one pose of the robot to the other [50], where the pose comprises the position and the orientation of the robot. An example of

a coverage route using Dubins curves to determine optimal deadheadings is given in Figure 6(a). A Reeds-Shepp model can also be used to determine such turns.

The Dubins curves can be inefficient when we have adjacent required edges as we may have to take extra turns to align the heading of the robot with the subsequent edge, as shown in Figure 6(b). Instead, we leverage the smooth turn model, described in the previous subsection, to allow the robots to deviate from their path within a given limit $\delta_{\max}$. This eliminates most of the extra turns generated by just using Dubins curves. An example is shown in Figure 6(c). In the case of fixed-wing UAVs that have a lower bound on the minimum speed, the algorithm selects Dubins curves for very sharp turns.

3) The MD-MEM-Turns Algorithm: Given an initial and a final heading angle at the depots, we require three kinds of cost functions for deadheading: (1) from a depot to any required arc $c(v_0, a)$, (2) from one required arc to another $c(a_1, a_2)$, and (3) from a required arc to a depot $c(a, v_0)$. Here, v_0 is a depot, and $a, a_1, a_2 \in \mathcal{A}_r$ are required arcs. For environments with cost functions that satisfy the triangle inequality, these cost functions can be computed in constant time using the above procedures for smooth turns and Dubins curves. When the triangle inequality is not satisfied, we use the technique of line graphs[3] [51], [52] to compute the shortest deadheading paths and cost functions, which can be computed in $\mathcal{O}(|E|^3)$ time, where E is the number of edges in the graph. The procedure MD-MEM-Turns::Initialize shows the modified initialization using multiple depots and turning costs. Similarly, the cost savings for merging can be modified to incorporate turning costs. The running time of the MEM algorithm does not change and is given by $\mathcal{O}(|V_d|m^2 \log m)$.

V. SIMULATIONS AND EXPERIMENTS

This section empirically demonstrates the high-quality solutions provided by the Merge-Embed-Merge (MEM) algorithm for the line coverage problem with multiple resource-constrained robots. We analyze the performance of the MEM

---

[3][51], [52] refer to line graphs as dual graphs, which should not be confused with dual graphs in graph theory.

(a) No adjacent edges, Dubins curves

(b) Adjacent edges, Dubins curves

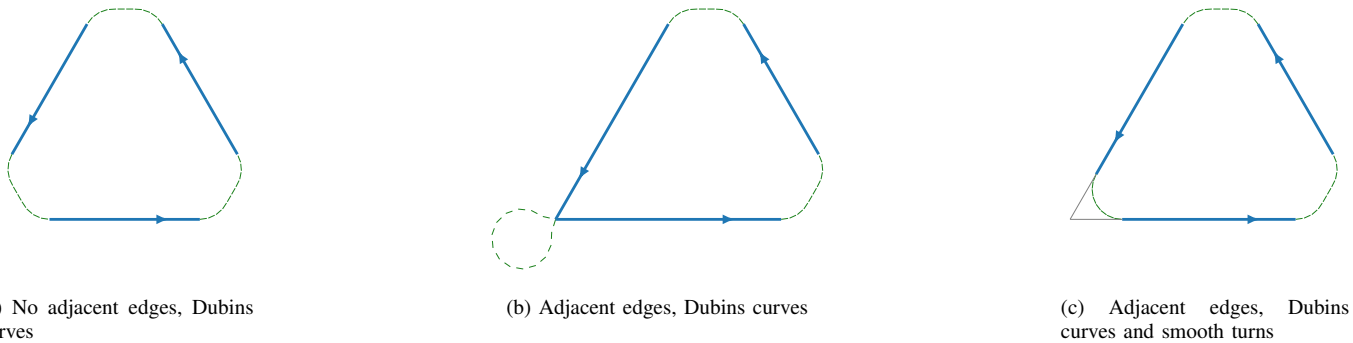(c) Adjacent edges, Dubins curves and smooth turns

Fig. 6. Modeling of deadheading paths for nonholonomic robots. (a) The graph does not contain sharp turns, and Dubins curves give optimal paths for deadheading. (b) When we have adjacent edges, common in road networks, Dubins curves can create several circular arcs to orient the robot along the edges. (c) We introduce smooth turns with Dubins curves to generate efficient deadheading paths between required edges.

Procedure MD-MEM-Turns::Initialize

Input : $G = (V, E, E_r)$, depots V_d, costs, demands, capacity Q

Output : Initialized coverage routes $\mathcal{R}$ with assigned depots

1   $\mathcal{R} \leftarrow \emptyset$; $k \leftarrow 1$;

2   **for** $e \in E_r$ **do**   // a_e and $\bar{a}_e$ are arcs for e

3     $R_k \leftarrow \emptyset$; $R_k.\text{cost} \leftarrow \infty$;

4   **for** $v \in V_d$ **do**   // Iterate over depots

5       $c \leftarrow c_d(v, a_e) + c_s(a_e) + c_d(a_e, v) + \lambda$;

6       $d \leftarrow q_d(v, a_e) + q_s(a_e) + q_d(a_e, v)$;

7   **if** $d \leq Q$ **and** $c < R_k.\text{cost}$ **then**

8         $R_k \leftarrow a_e$; $R_k.v_0 \leftarrow v$;   // Route with v_0

9       $\bar{c} \leftarrow c_d(v, \bar{a}_e) + c_s(\bar{a}_e) + c_d(\bar{a}_e, v) + \lambda$;

10      $\bar{d} \leftarrow q_d(v, \bar{a}_e) + q_s(\bar{a}_e) + q_d(\bar{a}_e, v)$;

11   **if** $\bar{d} \leq Q$ **and** $\bar{c} < R_k.\text{cost}$ **then**

12        $R_k \leftarrow \bar{a}_e$; $R_k.v_0 \leftarrow v$;   // Route with v_0

13     $\mathcal{R}.\text{push}(R_k)$; $k \leftarrow k + 1$;

---

algorithm in terms of computation time and the solution quality on a dataset of 50 road networks. We then demonstrate extensions of the MEM algorithm for large-scale graphs using a multi-depot formulation and for nonholonomic robots. We perform experiments with aerial robots on the UNC Charlotte campus road network. Finally, we demonstrate how the line coverage problem can be applied to the area coverage problem.

We use the DJI Phantom 4 quadrotor in our experiments, with the following cost model for traversing the edges. Denote the speed of the UAV by v and the wind speed by w. Let the travel vector $\mathbf{t}$ denote the traversal of an edge from its tail vertex v_t to the head vertex v_h. Let ϕ be the angle between the wind vector and the travel vector $\mathbf{t}$ for an edge. Then the effective speed of the UAV is given by:

$$v_{\text{eff}} = w \cos\phi + \sqrt{v^2 - w^2 \sin^2\phi}$$

The cost function is defined as the time taken for the UAV to traverse an edge:

$$c(v_t, v_h) = \frac{\|\mathbf{t}\|_2}{v_{\text{eff}}}$$

Here, $\|\mathbf{t}\|_2$ is the Euclidean distance from v_t to v_h. We use different speeds for servicing and deadheading, and the velocity v is set accordingly based on the travel mode. Note

that the cost function is direction-dependent due to wind, and hence, the graph is asymmetric.

For convenience, we use travel time as the cost and demand functions and specify the capacity in terms of the allowable flight time for the UAVs. These functions need not be the same in practice; instead, functions that model battery consumption can be used instead. Our open-source implementation allows for any non-negative edge cost and demand functions.

A. Analysis on Road Networks with Single Depot

We first perform a detailed simulation analysis for the single-depot line coverage problem to establish the efficacy and efficiency of the MEM algorithm. The analysis is performed on a dataset of 50 road networks, spanning about 1 km² area, from the most populous cities around the world, which we created using OpenStreetMap [54]. The road networks are representative of environments with linear features, and they provide widely varying graph structures for a thorough evaluation of the algorithm. These networks are represented by line segments, which form the set of required edges. The endpoints and the intersections of the road segments form the vertex set in the graph. For UAVs that fly at high altitudes, we add a non-required edge between each pair of vertices if a required edge does not exist between them.

The MEM algorithm is implemented in C++ and executed on a standard laptop with an Intel Core i7-1195G7 processor on a single core. We compare the quality of the solutions computed using the MEM algorithm with the solutions from solving the ILP formulation. The ILP formulation is solved using Gurobi [55], and the C++ API was used to interface with the solver. The ILP formulation, when executed on the laptop for an hour, is unable to compute a feasible solution for 41 of the 50 road networks. Thus, it is not very practical as the environment conditions can change, and efficient solutions may be needed quickly in response to emergency situations. Nevertheless, the ILP formulation provides an excellent benchmark to evaluate the quality of the heuristic solutions. We use the solutions obtained using the MEM algorithm to provide an initial solution to the solver, which helps in upper-bounding the branch-and-bound algorithm used by ILP solvers. Thus, the ILP solutions are always at least as good as the initial
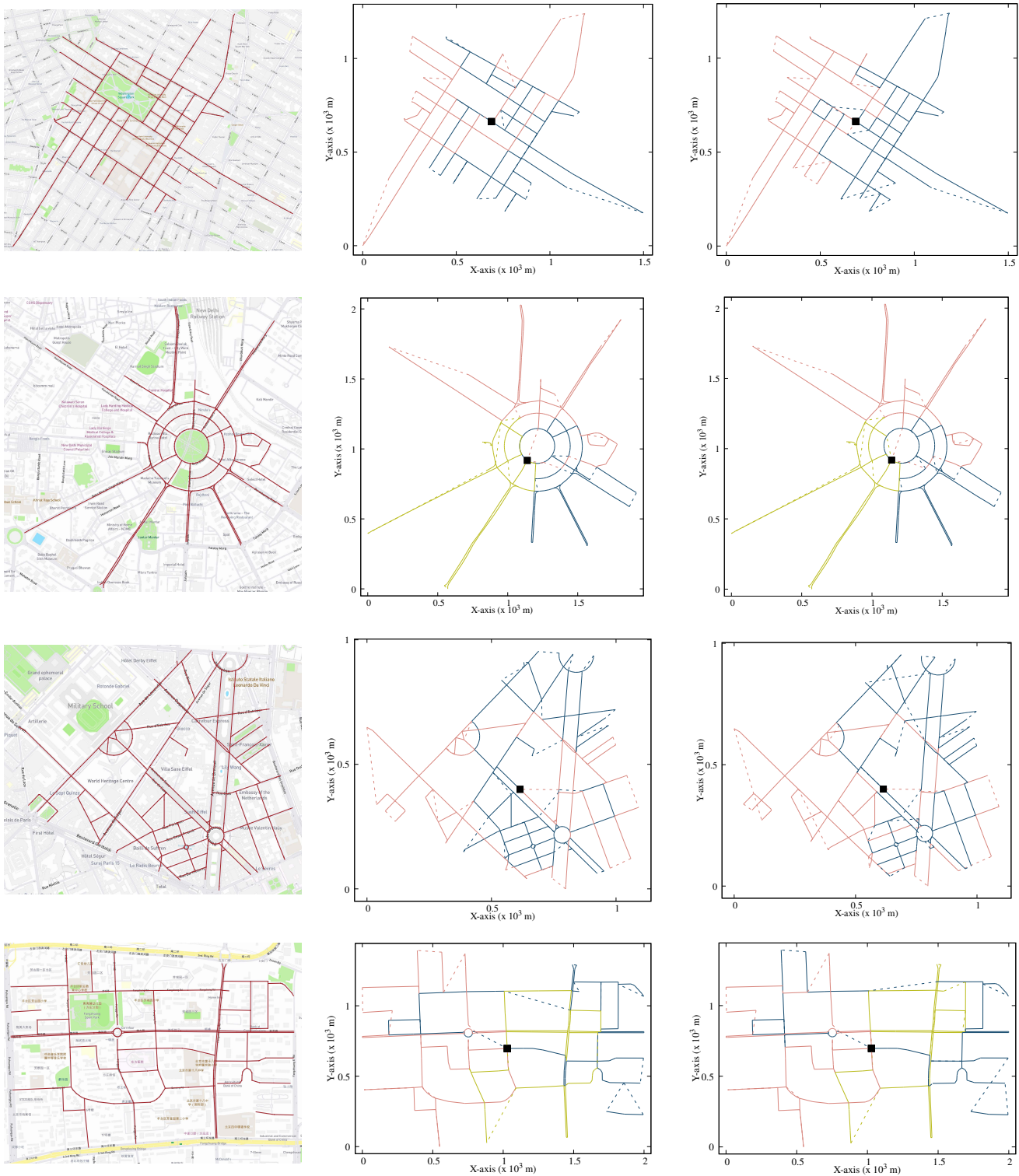
Fig. 7. Four of the 50 sample road networks obtained for the most populous cities: The first column is the map with the input graph, the second column is the solution obtained using the ILP formulation, and the third column is the solution obtained using the MEM algorithm. The road networks, from top to bottom, are from (a) New York, (b) Delhi, (c) Paris, and (d) Beijing. Only the required edges are shown in the input graph, and there is a required edge for each pair of vertices in the graph. For example, the New York graph has 379 vertices, 402 required edges, and 71,361 non-required edges. The solid lines represent servicing travel mode, and the dashed lines represent deadheading travel mode.
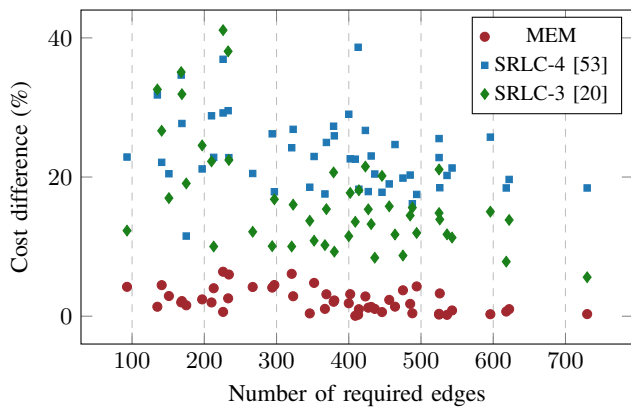
Fig. 8. Comparison of line coverage algorithms, computed as a cost difference percentage of the ILP solution. The MEM solutions, shown as red circles, are within 7% of the ILP solutions, and are significantly better than the 4-approx. [53] and the 3-approx. [20] algorithms for single robot line coverage without capacity constraints. The number of required edges corresponds to the number of line segments in the road networks.
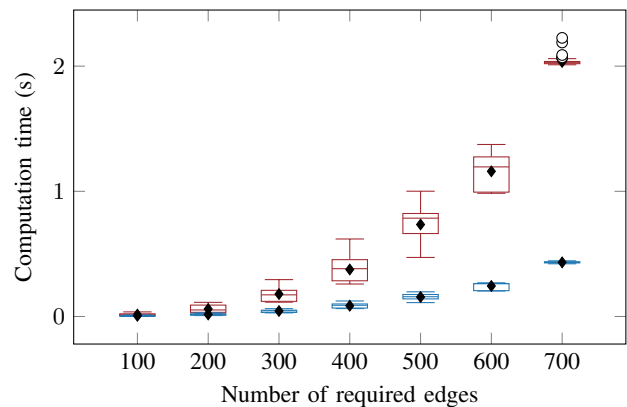

Fig. 9. Computation time for the MEM algorithm on the road network dataset. Each road network is placed in bins of size 100 based on the number of required edges. The MEM algorithm is executed 100 times for each road network. The results are shown as a boxplot: the red boxes represent the total computation time, while the blue boxes represent the time taken by the MEM algorithm. Diamond markers show the average, and the circles are outliers.

solutions. To obtain high-quality solutions, the ILP formulation is executed on a cluster node with an Intel Xeon Gold 6248R processor using 16 cores in parallel for each road network, and the execution time is limited to 24 hours.

We set the servicing and the deadheading speeds to $7\,\mathrm{m{\cdot}s^{-1}}$ and $10\,\mathrm{m{\cdot}s^{-1}}$, respectively. A wind of $2\,\mathrm{m{\cdot}s^{-1}}$ is simulated from the southwest direction, i.e., $\pi/4$ radians from the horizontal axis. For the first set of experiments, we set the capacity of the robot to 20 minutes $(1{,}200\,\mathrm{s})$. Figure 7 shows four of the fifty road networks, along with the routes obtained using the ILP formulation and the MEM algorithm. We consider a single depot location for these simulations, shown by a black square. The depot is set to be the vertex closest to the mean of all the vertices in the graph. The road networks provide a rich set of graph structures, as illustrated by the four sample graphs. The MEM solutions look similar to that of the ILP formulation, and the amount of deadheading is minimal.

Figure 8 presents the performance of the MEM algorithm, shown as red circles, in terms of cost difference percentage, computed as $100\frac{(c-c^*)}{c^*}$, where c and c^* are the costs of solutions computed using the algorithm and the ILP formulation, respectively. The solutions are compared with ones computed using the 4-approximation algorithm [53] and the 3-approximation algorithm [20] for the single robot line coverage. Even though the algorithms for the single robot case have theoretical bounds on the cost, they do not perform as well as the MEM algorithm for the road network dataset. Note that the single robot algorithms are solving a much simpler problem as they do not consider resource demands, capacity constraints, and depot locations. The difference in cost of the MEM solutions does not deviate significantly as the number of required edges increases. The performance of the ILP decreases rapidly for large graphs as the number of variables increases, and the ILP is not able to converge to an optimal solution within the computation time of 24 hours. Furthermore, graphs with a large network length require more routes, increasing the number of variables in the ILP formulation.

Figure 9 shows the computation time required to obtain solutions for the road network dataset. Most of the computation is spent processing the network and creating the graph. The largest road network with 730 required edges is solved in about $2\,\mathrm{s}$, of which the MEM algorithm takes less than $0.5\,\mathrm{s}$. For graphs with less than 200 required edges, the solutions are generated within $0.1\,\mathrm{s}$. The results indicate that the algorithm is very fast for robotics applications. It can also be used in real-time, where the graph structure and the remaining battery life are updated as robots traverse the environment.

Next, we analyze the performance of the MEM algorithm with different robot capacities. For each road network, the capacity is set as a fraction of the minimum cost required to cover the entire network using a single robot. Figure 10 shows the cost difference percentage between the solutions obtained using the ILP formulation and the MEM algorithm with varying capacity fractions. Observe that the cost difference percentage decreases as the capacity decreases, i.e., the performance of the MEM algorithm with respect to the ILP formulation improves. This happens because the number of routes K required to cover the entire road network increases as the capacity decreases, which, in turn, increases the number of variables in the ILP formulation. As a result, the ILP formulation is not able to compute near-optimal solutions within the computation time limit of 24 hours. In contrast, the performance of the MEM algorithm with respect to the ILP formulation, as given by the cost difference percentage, remains consistent for different capacities. Furthermore, the running time of the MEM algorithm is not significantly affected as it requires fewer merges as the capacity decreases.

The simulations on the road network dataset show that the MEM algorithm generates solutions with costs comparable to that of the bounded-computation-time ILP formulation, i.e., within 7%, and with a maximum computation time of around $2\,\mathrm{s}$. These results show that the algorithm is suitable for deploying robots, in particular aerial robots, for coverage of linear infrastructure.
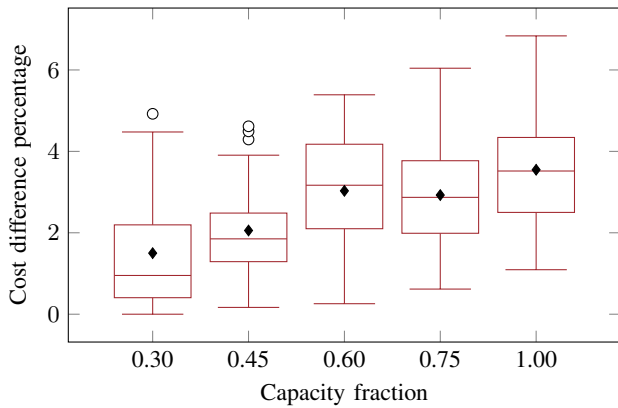
Fig. 10. The cost difference between the solutions generated using the MEM algorithm and the ILP solution for different capacities. For each instance, the capacity is set as a fraction of the route cost for a single robot with infinite capacity. In the boxplots, circles show the outliers, and the diamond markers show the average. The average difference in the cost between the solutions generated by the MEM algorithm and the ILP formulation is 2.61%.

Fig. 11. Performance analysis of the MD-MEM algorithm for large road networks spanning $3\,\text{km} \times 3\,\text{km}$ area with multiple depots. The MD-MEM solutions, shown as red circles, are consistently better than the 4-approx. [53] and the 3-approx. [20] algorithms for single robot line coverage without capacity constraints.

B. Analysis on 50 Large Road Networks with Multiple Depots

We present simulation analysis for 50 large road networks, each spanning a $3\,\text{km} \times 3\,\text{km}$ area. The number of required edges ranges from 481 to 4831, and the network length from $22\,\text{km}$ to $108\,\text{km}$. The robot and environment parameters are the same as in the previous section. Since the road networks are large, it is not possible to service the entire network from a single depot. Thus, we use the multi-depot approach to generate line coverage routes, as discussed in Section IV-B. Depot locations are generally selected based on the ease of field operation of robots. In our simulations, the depot locations are randomly selected from the vertices of the required edges of the road network. The number of depots is set based on the total average demand as follows:

$$\text{number of depots} = 1 + \left\lceil \sum_{e \in E_r} \frac{q_s(a_e) + q_s(\bar{a}_e)}{2Q} \right\rceil. \quad (20)$$

We use the multi-depot merge-embed-merge (MD-MEM) to generate solutions. Figure 12 shows routes for three road networks. Due to the large number of edges, it is not possible to solve the problem using the ILP formulation. Instead, we compare the MD-MEM algorithm with the SRLC-4-approximation [53] and the SRLC-3-approximation [20] algorithms for the single robot line coverage problem (SRLC). These two single-robot algorithms are solving a simpler version of the problem as they do not consider capacity constraints and depot locations. Yet, the MD-MEM algorithm performs significantly better than the single robot algorithms, as shown in Figure 11. In the figure, the cost difference percentage is computed with respect to the total average cost as follows:

$$\text{cost difference } \% = 100 \times \frac{c - \bar{c}}{\bar{c}}, \quad \bar{c} = \sum_{e \in E_r} \frac{c_s(a_e) + c_s(\bar{a}_e)}{2}.$$

In prior work, the multi-depot problem was solved by clustering the edges based on the depot locations as a preprocessing step [28], [3]. These cl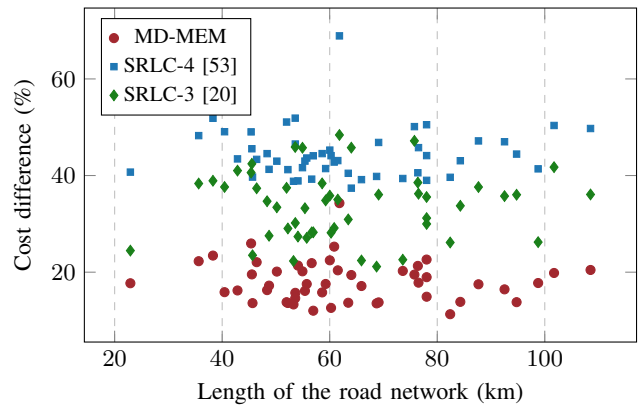usters form a set of subgraphs, and each subgraph is then solved as a single-depot problem. However, this process yields a larger number of routes, as routes that service edges from different subgraphs are not permitted, thereby restricting the solution space. In contrast, the MD-MEM algorithm considers the depots within the routing process itself, resulting in a smaller number of efficient routes. Figure 13 shows a comparison of the number of routes generated using the clustering approach to create subgraphs and the MD-MEM algorithm—it can be seen that the MD-MEM algorithm consistently generates fewer routes. As the number of routes corresponds to the number of flights, a reduction in the number of routes can impact the operation costs significantly in terms of equipment and operators required. In terms of total cost, the MD-MEM consistently performs better than the clustering approach, with an average cost difference of 2.5%. The cost difference is not huge as both methods use the same MEM structure to generate routes; the primary advantage is in the reduction of the number of routes.

Figure 14 shows the computation time for the MD-MEM algorithm on the large road networks dataset. The computation time is averaged over 100 runs. Only the time required by the MD-MEM algorithm is considered, and the time for loading the dataset is not included. The standard deviation is very low (less than 2 s) and is not visible in the figure. The computation time for the largest network consisting of 4831 required edges is within 100 s. These experiments establish the reliability of the MD-MEM algorithm to generate high-quality solutions with low computation effort for large line coverage networks.

C. UAV Experiment on a Road Network with a Single Depot

We performed experiments with a UAV on a portion of the UNC Charlotte campus road network, shown in Figure 15. The service and the deadhead speed were set to $3.33\,\text{m·s}^{-1}$ and $5\,\text{m·s}^{-1}$, and a wind speed of $0.89\,\text{m·s}^{-1}$ from the west was incorporated based on the wind conditions on the day of the experiment. A conservative capacity of $600\,\text{s}$ was selected. Coverage routes were generated using the ILP formulation and the MEM algorithm. The ILP formulation gave the optimal
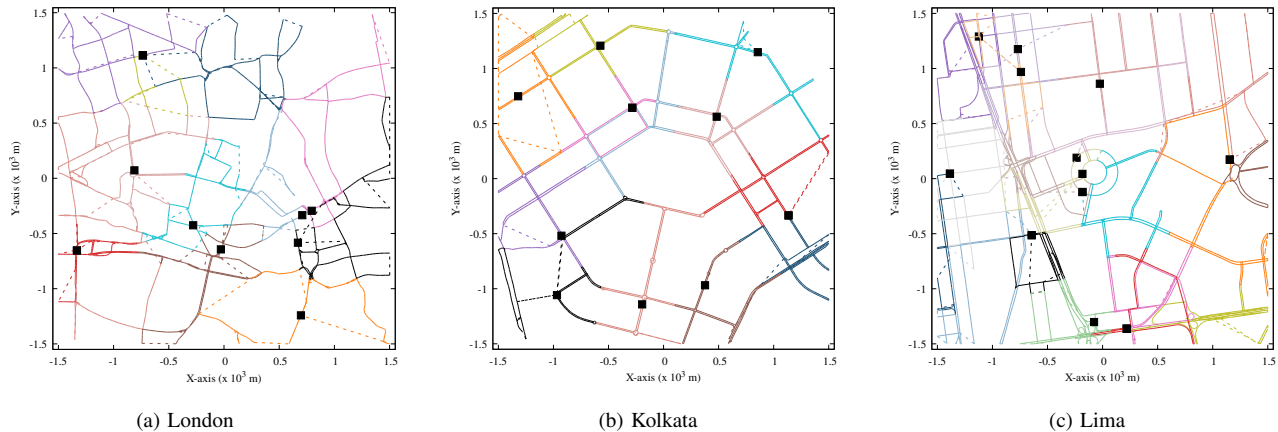
This article has been accepted for publication in IEEE Transactions on Robotics. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TRO.2024.3355802

16

(a) London   (b) Kolkata   (c) Lima

Fig. 12. Line coverage solutions for large road networks, each spanning a $3\,\mathrm{km} \times 3\,\mathrm{km}$ area. The solutions are generated using the multi-depot merge-embed-merge (MD-MEM) algorithm. (a) The London road network has the largest number of required edges in the dataset, with 4831 required edges, over 10 million non-required edges, and about $60\,\mathrm{km}$ total network length. (b) The Kolkata road network has 2160 required edges and has a total network length of $66\,\mathrm{km}$. (c) With $108\,\mathrm{km}$ of total network length, the Lima road network is the longest. The depot locations are randomly selected from the vertices of the required edges of the road network. The MD-MEM algorithm handles multiple depots within the routing framework.
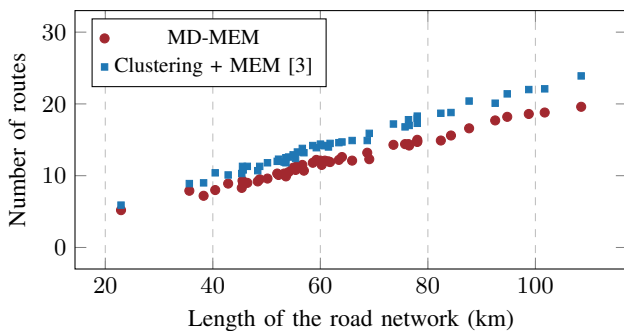

Fig. 13. Comparison of the number of routes generated using the clustering approach to generate subgraphs [3] and the MD-MEM algorithm. The depot locations are selected at random, and the number of routes is the average over 10 runs. The clustering approach generates subgraphs based on the depot locations and solves each subgraph as a single-depot problem. The MD-MEM algorithm consistently generates fewer routes than the clustering approach.
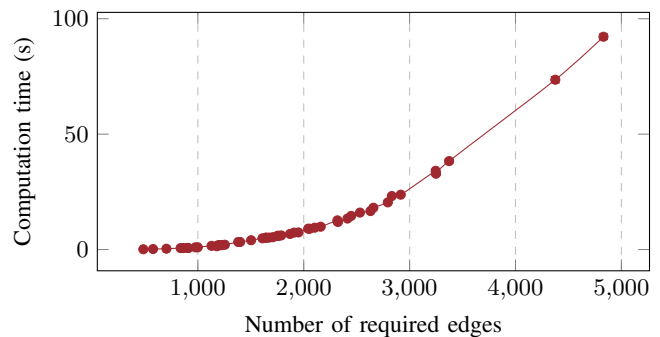

Fig. 14. Computation time of the MD-MEM algorithm for very large graphs, averaged over 100 runs. For the largest graph with 4831 required edges, the algorithm can generate solutions within $100\,\mathrm{s}$.

TABLE II
COMPARISON OF COMPUTED AND ACTUAL FLIGHT TIMES

	Computed Cost (s)		Actual Flight Time (s)	
	Route 1	Route 2	Route 1	Route 2
ILP	462	554	527	642
MEM	473	599	548	688

solutions. The computed and the actual flight times are shown in Table II. The actual flight times include take-off and landing and are close to the computed cost.

The experiment shows that the line coverage problem is well-suited to model coverage of linear infrastructure such as a road network. The problem models two different modes of travel and incorporates wind conditions in the formulation. Furthermore, activating the sensors only during servicing helps in reducing the amount of data that requires post-processing.

D. Experiment on a Large-Scale Road Network

We used the multi-depot MEM algorithm for the UNC Charlotte campus road network spanning an area of $1.5\,\mathrm{km}^2$ and a length of $12\,\mathrm{km}$, shown in Figure 1. The road network consists of 842 vertices, 865 required edges, and 353,196 non-required edges. k-medoids clustering was used to obtain eight depot locations, shown as black squares in the figure. The depots are well distributed in the road network such that no road segment is far away from all the depots. The multi-depot MEM algorithm computed eight routes, and no two routes share the same depot. Note that the k-medoids algorithm is only used to compute the depot locations, and we do not cluster the edges. Service and deadhead speeds of $5\,\mathrm{m \cdot s^{-1}}$ and $8\,\mathrm{m \cdot s^{-1}}$ were set for the experiments. The computed routes and the orthomosaic generated from the images collected during flights are shown in Figure 1. Table III gives computed costs, actual flight times, and the number of images collected for each route. For these experiments, we did not incorporate the wind conditions, which could be a reason for a higher deviation between the computed cost and the actual flight time.

It can be observed from the routes that the individual routes

(a) Input Road Network

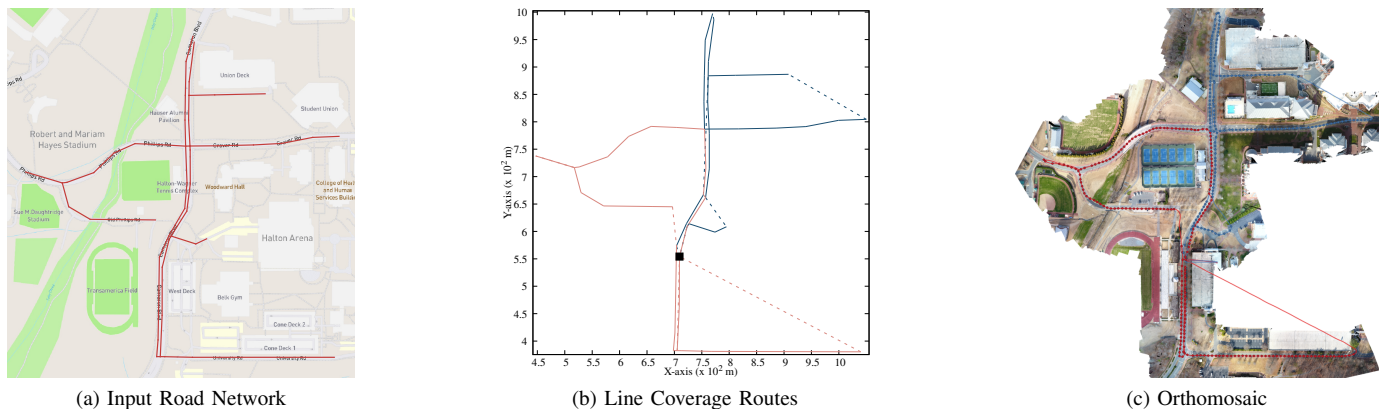(b) Line Coverage Routes

(c) Orthomosaic

Fig. 15. Line coverage of a portion of the UNC Charlotte road network. (a) The input road network has a length of 2,658 m with 48 vertices and 48 required edges (shown as red lines). There are 1,128 non-required edges formed by pairs of vertices (not shown). (b) Two routes distinguished by different colors are computed using the MEM algorithm. The algorithm computes a depot location, shown by the black square, from where the UAVs start and end their routes. The solid lines represent servicing, while the dashed lines represent deadheading. (c) An orthomosaic generated from the images taken by the UAVs flown autonomously along the computed routes. The lines show the actual flight path, and the dots are the locations where the images were taken.

TABLE III
DATA OF FLIGHTS FOR THE UNC CHARLOTTE ROAD NETWORK

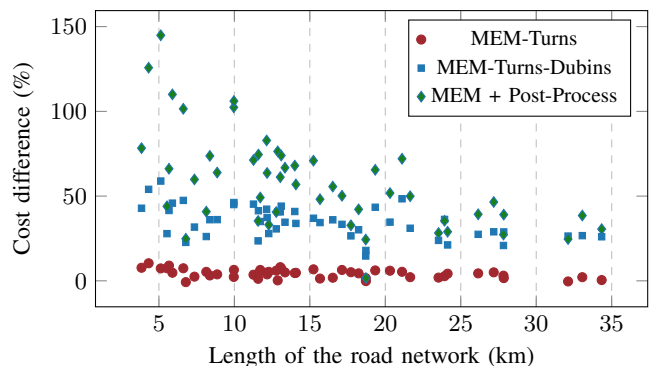| Computed Cost (s)	Flight time (s)	Number of images
215	346	87
458	616	180
407	496	170
538	657	221
371	493	140
449	599	192
481	627	210
277	393	120

Fig. 16. Performance analysis of the MEM algorithm for nonholonomic robots on the smaller road network dataset with a single depot. The cost difference is measured with respect to MEM solutions without nonholonomic constraints. The green diamonds correspond to the post-processing of holonomic routes to satisfy the nonholonomic constraints. The MD-MEM-Turns algorithm is executed with only Dubins curves (blue squares) and with both Dubins curves and smooth turns (red circles). Incorporating both Dubins curves and smooth turns in the MEM algorithm consistently computes lowest cost solutions.

primarily consist of road segments that are close to each other and are connected, indicating that the MEM algorithm can efficiently distribute the line features among routes. Furthermore, the routes are assigned to the closest depot, showing that the multiple depot formulation of the MEM algorithm is well-suited for the line coverage problem with large graphs.

E. Nonholonomic Robots

We analyzed the performance of the MD-MEM-Turns algorithm for nonholonomic robots on the smaller 50 road network dataset with a single depot. The service and the deadhead speeds were set to $3.33 \, \text{m·s}^{-1}$ and $5.00 \, \text{m·s}^{-1}$. The maximum angular velocity and acceleration were set to $\pi/4 \, \text{rad·s}^{-1}$ and $3.00 \, \text{m·s}^{-2}$, and the flight time was limited to 1800 s. A deviation limit $\delta_{\max}$ of 2 m was selected to allow smooth turns for adjacent required edges. The results, shown in Figure 16, indicate that incorporating smooth turns (red circles) consistently computes solutions with significantly lower costs than just using Dubins curves (blue squares). The post-processing approach (green diamonds) does not perform as well as the MD-MEM-Turns algorithm, as it does not consider the turning costs during the route generation process. Furthermore, it can also violate the capacity constraints. When using only Dubins curves with the MEM algorithm, the average improvement over the post-processing approach is 13%, whereas using both Dubins curves and smooth turns gives an improvement of 32%.

Figure 17 shows a network of lanes on a set of parking lots as an illustrative example. Two routes that respect the nonholonomic constraints are generated using the MD-MEM-Turns algorithm with two depots and a flight time limit of 600 s. The deadheadings are composed of Dubins curves and smooth turns for non-adjacent and adjacent required edges, respectively. Note that the algorithm computed routes that cover separated regions while reducing the amount of deadheading, thereby optimizing the total cost of the routes. The computation time of the MD-MEM-turns algorithm is similar to that of the standard MEM algorithm because of the constant time computation of savings, as discussed in Section IV-C.

F. Area Coverage: Multiple Depots and Nonholonomic Robots

We now illustrate the use of the algorithms developed for the line coverage problem to generate routes for area coverage. Given a region $\mathcal{R} \subset \mathbb{R}^2$, the area coverage problem is to find

(a) Input Road Network
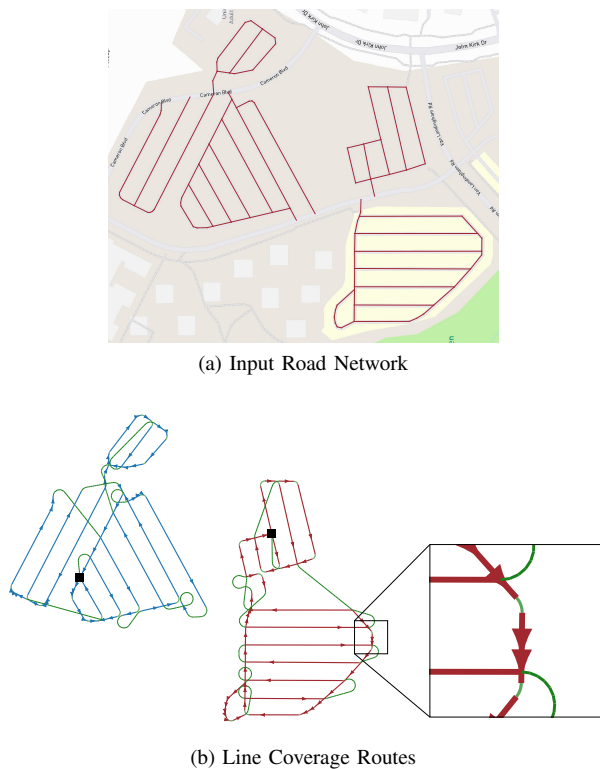

(b) Line Coverage Routes

Fig. 17. Line coverage of a network of lanes in a set of parking lots: (a) The total length of the lanes in the input graph is 2,982 m. There are 90 vertices, 104 required edges, and 4,005 non-required edges. (b) Coverage routes using two depots and nonholonomic robots. There are two routes distinguished by different colors. The green lines show deadheadings composed of Dubins curves and smooth turns. The arrows indicate the direction of travel. The inset shows an enlarged view of smooth turns for adjacent required edges.

a set of routes for a team of robots such that the total cost of the routes is minimized, and the robots service all the points in the region. The sensor field-of-view is an important parameter that needs to be factored in when determining the routes. We developed a procedure to transform the area coverage problem to the line coverage problem and then used the MEM algorithm to generate efficient routes that consider resource constraints and asymmetric costs [41].

The formulation for solving the area coverage problem consists primarily of three components: (1) *Cell decomposition* of the environment, (2) *Service track generation* for individual cells, and (3) *Routing* to traverse the service tracks. The service tracks form the linear features that robots must service. The problem is formulated as a graph, and efficient line coverage algorithms are then used to generate routes that traverse the service tracks. Moreover, the transformation facilitates a significant generalization of the cell decomposition component to reduce the number of turns the robots must take. In particular, the cells are no longer required to be *monotone* polygons [56] with respect to the service direction. This generalization enables additional service directions for the cells to minimize the number of turns and reduces the number of service tracks by avoiding overlapping sensor coverage regions at the common boundary of adjacent cells. It was established in [41] that the above approach, aided by the line coverage algorithms, gives solutions that are better than other approaches in the literature.

The line coverage algorithms for multiple depots and non-holonomic constraints extend to the area coverage problem. Figure 18 illustrates solutions computed using the MD-MEM-Turns algorithm in an outdoor environment setting from [41]. The environment spans a $19,000\,\mathrm{m}^2$ region with a building containing 45 vertices. As UAVs can fly at high altitudes, we allow non-required edges that cross the building. However, only the region surrounding the building needs to be covered. Figure 18(a) shows the cell decomposition of the region into five cells. The service tracks, shown in Figure 18(b), take the field-of-view of the sensor into account. These service tracks form the linear features, i.e., the required edges, for the line coverage problem. Dubins curves are used to deadhead between pairs of non-adjacent required edges, and smooth turns are used to deadhead between adjacent required edges. Figure 18(c) shows the two routes computed using the MEM algorithm with multiple depots and nonholonomic constraints. The example illustrates the use of line coverage algorithms for area coverage, and shows that the enhancements to algorithms for line coverage translate directly to area coverage.

VI. CONCLUSION

The paper presented the line coverage problem with multiple resource-constrained robots for coverage of linear features such as road networks and power lines. The environment was modeled as a graph with the linear features as a set of required edges. The formulation permits non-required edges that can enable faster travel, but the robots need not cover them. The edges have non-negative costs (e.g., travel time) and resource demands (e.g., battery life), which can be direction dependent to facilitate the modeling of wind conditions, uneven terrain, and one-way streets. The goal is to compute routes that together cover all the required edges while minimizing the total cost of the routes under the constraint that each route's total demand is within the robot's resource capacity.

We posed line coverage as an optimization problem on graphs and formulated it as an integer linear program (ILP). As the problem is NP-hard, we designed a heuristic algorithm, Merge-Embed-Merge (MEM), with a time complexity of $\mathcal{O}(m^2 \log m)$, where m is the number of linear features. Leveraging the constructive nature of the MEM algorithm, we extended the algorithm to multiple variants of the line coverage problem. To address large graphs, we formulated the line coverage problem with multiple depots and designed the Multi-Depot MEM (MD-MEM) algorithm. We incorporated smooth turns and nonholonomic constraints, resulting in the MD-MEM-Turns algorithm. These are the first set of polynomial-time algorithms that directly incorporate multiple depots, smooth turns, and nonholonomic constraints into the line coverage problem with resource-constrained robots.

The algorithms were evaluated on medium and large road networks from the world's 50 most populous cities, which showed that the MEM algorithm computes high-quality solutions with costs within 7% of the ILP solutions. The MEM algorithm also performs similarly when the capacity of the robots is varied. The computation times show that the algorithm is very fast, solving single-depot road networks within

(a) Cell decomposition   (b) Service tracks   (c) Routes with nonholonomic robots
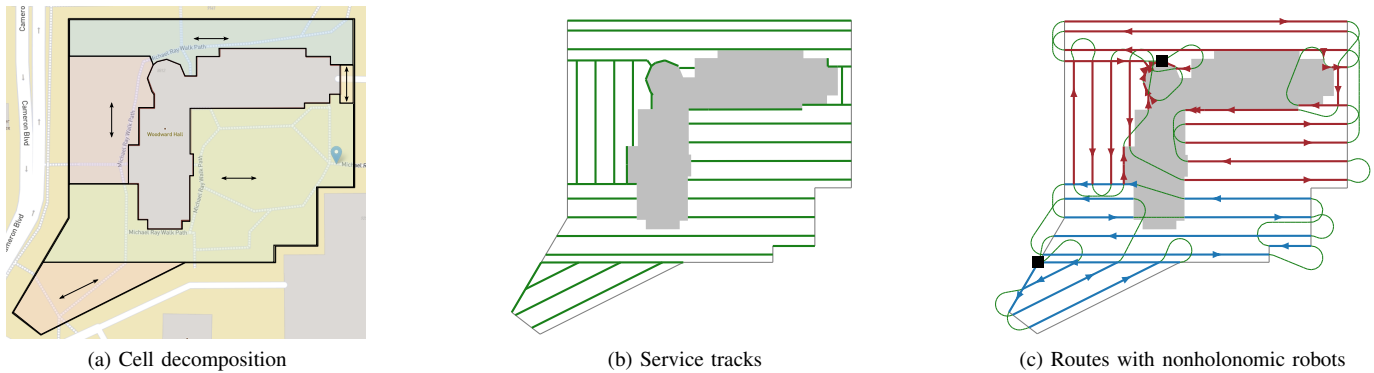
Fig. 18. Area coverage using aerial robots for a real-world example from [41]: (a) The region surrounding a building needs to be covered. The cell decomposition is shown with double-head arrows indicating service directions. (b) Service tracks are generated for each cell independently. The tracks are parallel to the service directions obtained from cell decomposition. These tracks form the linear features for the line coverage problem. (c) Two routes computed by the MEM algorithm for multiple depots and nonholonomic robots. The green lines indicate deadheading and comprise Dubins curves between non-adjacent tracks and smooth turns between adjacent tracks.

a fraction of a second. On large road networks, the MD-MEM algorithm generates solutions that are significantly better than the existing approximation algorithms for the single robot line coverage problem. When compared to existing clustering-based approaches, the MD-MEM algorithm consistently generates fewer routes, thereby optimizing operating costs.

The algorithms were demonstrated in physical experiments on the UNC Charlotte road network. For the first experiment, two routes from a single depot location were autonomously executed by a commercial UAV to cover a portion of the road network. In the second experiment, the UAVs executed routes from the multiple depot formulation for a large road network.

We showed that the new capabilities of incorporating multiple depots and nonholonomic robots translate directly to routing for the area coverage problem. It is easy to transform point features to degenerate linear features by creating artificial edges. Thus, all three types of features—points, lines or curves, and areas—can be modeled within the same *generalized coverage* [57] framework.

Future work includes addressing line coverage with a heterogeneous team of robots that vary in their resource capacity and sensor capabilities. We are also interested in exploring the orienteering routing problem in the context of linear features, where the features have associated profits, and the objective is to service edges that maximize the total profit. Since the MEM algorithm is very fast and can potentially be used for computing routes online on the robots, we plan to characterize the performance of the algorithm in distributed and online settings. From a theoretical standpoint, establishing an approximation guarantee or proving the inapproximability of the problem is an important direction for future work.

ACKNOWLEDGMENTS

REFERENCES

[1] A. Corberán and G. Laporte, Eds., *Arc routing: problems, methods, and applications*. Philadelphia, PA: Society for Industrial and Applied Mathematics (SIAM), 2014.

[2] A. Corberán, R. Eglese, G. Hasle, I. Plana, and J. M. Sanchis, "Arc routing problems: A review of the past, present, and future," *Networks*, vol. 77, no. 1, pp. 88–115, 2021.

[3] S. Agarwal and S. Akella, "Line coverage with multiple robots," in *IEEE International Conference on Robotics and Automation (ICRA)*, Paris, France, 2020, pp. 3248–3254.

[4] T. Roughgarden, *Algorithms Illuminated, Part 4: Algorithms for NP-Hard Problems*. New York, USA: Soundlikeyourself Publishing, 2020.

[5] R. M. Karp, "Reducibility among combinatorial problems," in *Complexity of Computer Computations: Proceedings of a symposium on the Complexity of Computer Computations*, R. E. Miller, J. W. Thatcher, and J. D. Bohlinger, Eds. Boston, MA: Springer US, 1972, pp. 85–103.

[6] N. Christofides, "Worst-case analysis of a new heuristic for the travelling salesman problem," Graduate School of Industrial Administration, Carnegie Mellon University, Pittsburgh, PA, Tech. Rep. 388, 1976.

[7] V. Traub and J. Vygen, "An improved approximation algorithm for ATSP," in *52nd Annual ACM SIGACT Symposium on Theory of Computing*, ser. STOC 2020, Chicago, IL, USA, 2020, pp. 1–13.

[8] B. L. Golden, L. Levy, and R. Vohra, "The orienteering problem," *Naval Research Logistics (NRL)*, vol. 34, no. 3, pp. 307–318, 1987.

[9] P. Vansteenwegen, W. Souffriau, and D. V. Oudheusden, "The orienteering problem: A survey," *European Journal of Operational Research*, vol. 209, no. 1, pp. 1–10, 2011.

[10] C. Chekuri, N. Korula, and M. Pál, "Improved Algorithms for Orienteering and Related Problems," *ACM Transactions on Algorithms (TALG)*, vol. 8, no. 3, pp. 1–27, 2012.

[11] W. Xu, W. Liang, Z. Xu, J. Peng, D. Peng, T. Liu, X. Jia, and S. K. Das, "Approximation algorithms for the generalized team orienteering problem and its applications," *IEEE/ACM Transactions on Networking*, vol. 29, no. 1, pp. 176–189, Feb 2021.

[12] R. Pěnička, J. Faigl, P. Váňa, and M. Saska, "Dubins orienteering problem," *IEEE Robotics and Automation Letters*, vol. 2, no. 2, pp. 1210–1217, April 2017.

[13] J. Faigl, P. Váňa, and R. Pěnička, "Multi-vehicle close enough orienteering problem with Bézier curves for multi-rotor aerial vehicles," in *2019 International Conference on Robotics and Automation (ICRA)*, May 2019, pp. 3039–3044.

[14] S. Agarwal and S. Akella, "The correlated arc orienteering problem," in *Algorithmic Foundations of Robotics XV*, S. M. LaValle, J. M. O'Kane, M. Otte, D. Sadigh, and P. Tokekar, Eds. Cham, Germany: Springer International Publishing, 2023, pp. 402–418.

[15] P. Toth and D. Vigo, *Vehicle Routing*, 2nd ed., D. Vigo and P. Toth, Eds. Philadelphia: Society for Industrial and Applied Mathematics, 2014.

[16] S. Khuller, A. Malekian, and J. Mestre, "To fill or not to fill: The gas station problem," in *Algorithms – ESA 2007*, L. Arge, M. Hoffmann, and E. Welzl, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2007, pp. 534–545.

This article has been accepted for publication in IEEE Transactions on Robotics. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TRO.2024.3355802

20

[17] K. Sundar and S. Rathinam, "Algorithms for routing an unmanned aerial vehicle in the presence of refueling depots," *IEEE Transactions on Automation Science and Engineering*, vol. 11, no. 1, pp. 287–294, 2014.

[18] J. Edmonds and E. L. Johnson, "Matching, Euler tours and the Chinese postman," *Mathematical Programming*, vol. 5, no. 1, pp. 88–124, Dec. 1973.

[19] G. N. Frederickson, "Approximation algorithms for some postman problems," *Journal of the ACM*, vol. 26, no. 3, pp. 538–554, Jul. 1979.

[20] S. Agarwal and S. Akella, "Approximation algorithms for the single robot line coverage problem," in *Algorithmic Foundations of Robotics XIV*, S. M. LaValle, M. Lin, T. Ojala, D. Shell, and J. Yu, Eds. Cham, Germany: Springer International Publishing, 2021, pp. 534–550.

[21] ——, "The single robot line coverage problem: Theory, algorithms, and experiments," *Networks*, vol. 82, no. 4, pp. 479–505, 2023.

[22] B. Boyacı, T. H. Dang, and A. N. Letchford, "Improving a constructive heuristic for the general routing problem," *Networks*, vol. 81, no. 1, pp. 93–106, 2023.

[23] K. Easton and J. Burdick, "A coverage algorithm for multi-robot boundary inspection," in *IEEE International Conference on Robotics and Automation*, Barcelona, Spain, Apr. 2005, pp. 727–734.

[24] B. L. Golden and R. T. Wong, "Capacitated arc routing problems," *Networks*, vol. 11, no. 3, pp. 305–315, 1981.

[25] S. Wøhlk, "An Approximation Algorithm for the Capacitated Arc Routing Problem," *The Open Operational Research Journal*, vol. 2, no. 1, pp. 8–12, Oct. 2008.

[26] L. Gouveia, M. C. Mourão, and L. S. Pinto, "Lower bounds for the mixed capacitated arc routing problem," *Computers & Operations Research*, vol. 37, no. 4, pp. 692–699, 2010.

[27] G. Kirlik and A. Sipahioglu, "Capacitated arc routing problem with deadheading demands," *Computers & Operations Research*, vol. 39, no. 10, pp. 2380–2394, 2012.

[28] L. Muyldermans, D. Cattrysse, and D. V. Oudheusden, "District design for arc-routing applications," *Journal of the Operational Research Society*, vol. 54, no. 11, pp. 1209–1221, 2003.

[29] M. C. Mourão, A. C. Nunes, and C. Prins, "Heuristic methods for the sectoring arc routing problem," *European Journal of Operational Research*, vol. 196, no. 3, pp. 856–868, 2009.

[30] A. Corberán and C. Prins, "Recent results on arc routing problems: An annotated bibliography," *Networks*, vol. 56, no. 1, pp. 50–69, 2010.

[31] M. Dille and S. Singh, "Efficient aerial coverage search in road networks," in *AIAA Guidance, Navigation, and Control Conference*, Boston, MA, USA, Aug. 2013, pp. 5048–5067.

[32] H. Oh, S. Kim, A. Tsourdos, and B. White, "Coordinated road-network search route planning by a team of UAVs," *International Journal of Systems Science*, vol. 45, no. 5, pp. 825–840, 2014.

[33] K. Williams and J. Burdick, "Multi-robot boundary coverage with plan revision," in *IEEE International Conference on Robotics and Automation*, Orlando, USA, 2006, pp. 1716–1723.

[34] L. Xu and T. Stentz, "A fast traversal heuristic and optimal algorithm for effective environmental coverage," in *Proceedings of Robotics: Science and Systems*, Zaragoza, Spain, June 2010, pp. 161–168.

[35] L. Xu and A. Stentz, "An efficient algorithm for environmental coverage with multiple robots," in *IEEE International Conference on Robotics and Automation (ICRA)*, Shanghai, China, May 2011, pp. 4950–4955.

[36] J. F. Campbell, A. Corberán, I. Plana, and J. M. Sanchis, "Drone arc routing problems," *Networks*, vol. 72, no. 4, pp. 543–559, 2018.

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

[38] R. Mannadiar and I. Rekleitis, "Optimal coverage of a known arbitrary environment," in *IEEE International Conference on Robotics and Automation (ICRA)*, Anchorage, USA, 2010, pp. 5525–5530.

[39] N. Karapetyan, K. Benson, C. McKinney, P. Taslakian, and I. Rekleitis, "Efficient multi-robot coverage of a known environment," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Vancouver, Canada, 2017, pp. 1846–1852.

[40] G. N. Frederickson, M. S. Hecht, and C. E. Kim, "Approximation algorithms for some routing problems," in *17th Annual Symposium on Foundations of Computer Science*, Houston, USA, 1976, pp. 216–227.

[41] S. Agarwal and S. Akella, "Area Coverage With Multiple Capacity-Constrained Robots," *IEEE Robotics and Automation Letters*, vol. 7, no. 2, pp. 3734–3741, 2022.

[42] Yongguo Mei, Yung-Hsiang Lu, Y. C. Hu, and C. S. G. Lee, "Deployment of mobile robots with energy and timing constraints," *IEEE Transactions on Robotics*, vol. 22, no. 3, pp. 507–522, 2006.

[43] C. D. Franco and G. C. Buttazzo, "Energy-aware coverage path planning of UAVs," in *IEEE International Conference on Autonomous Robot Systems and Competitions*, Vila Real, Portugal, 2015, pp. 111–117.

[44] G. Clarke and J. W. Wright, "Scheduling of vehicles from a central depot to a number of delivery points," *Operations Research*, vol. 12, no. 4, pp. 568–581, 1964.

[45] B. L. Golden, J. S. Dearmon, and E. K. Baker, "Computational experiments with algorithms for a class of routing problems," *Computers & Operations Research*, vol. 10, no. 1, pp. 47–59, 1983.

[46] S. Dasgupta, C. Papadimitriou, and U. Vazirani, *Algorithms*. McGraw-Hill Higher Education New York, 2006.

[47] K. Yang and S. Sukkarieh, "An analytical continuous-curvature path-smoothing algorithm," *IEEE Transactions on Robotics*, vol. 26, no. 3, pp. 561–568, 2010.

[48] A. Ravankar, A. A. Ravankar, Y. Kobayashi, Y. Hoshino, and C.-C. Peng, "Path Smoothing Techniques in Robot Navigation: State-of-the-Art, Current and Future Challenges," *Sensors*, vol. 18, no. 9, 2018.

[49] K. M. Lynch and F. C. Park, *Modern Robotics: Mechanics, Planning, and Control*. USA: Cambridge University Press, 2017.

[50] S. LaValle, *Planning Algorithms*. Cambridge University Press, 2006.

[51] S. Winter, "Modeling costs of turns in route planning," *GeoInformatica*, vol. 6, no. 4, pp. 345–361, 2002.

[52] R. Geisberger and C. Vetter, "Efficient Routing in Road Networks with Turn Costs," in *Experimental Algorithms*, P. M. Pardalos and S. Rebennack, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 100–111.

[53] R. van Bevern, C. Komusiewicz, and M. Sorge, "A parameterized approximation algorithm for the mixed and windy capacitated arc routing problem: Theory and experiments," *Networks*, vol. 70, no. 3, pp. 262–278, Oct. 2017.

[54] OpenStreetMap contributors, "Planet dump retrieved from https://planet.osm.org," Online: https://www.openstreetmap.org, 2022.

[55] Gurobi Optimization, LLC, "Gurobi optimizer reference manual," 2021. [Online]. Available: http://www.gurobi.com

[56] M. de Berg, O. Cheong, M. van Kreveld, and M. Overmars, *Computational Geometry: Algorithms and Applications*, 3rd ed. Berlin: Springer-Verlag, 2008.

[57] S. Agarwal, "Generalized coverage with multiple robots: Theory, algorithms, and experiments," Ph.D. dissertation, University of North Carolina at Charlotte, 2022.

Saurav Agarwal is a postdoctoral fellow at the GRASP Lab, University of Pennsylvania. His research interests include graph neural networks, combinatorial optimization, and approximation algorithms with applications to motion planning and control of multi-robot systems. He received his Ph.D. in Computer Science from the University of North Carolina at Charlotte, where he introduced the generalized coverage problem, designed efficient algorithms with provable guarantees for multi-robot path planning problems, and conducted experiments with aerial and ground robots. Previously, at the Indian Institute of Technology, Madras, he worked on kinematics, design, and control of mechanisms and parallel robots.

Srinivas Akella is a Professor of Computer Science at the University of North Carolina at Charlotte. He received his Ph.D. in Robotics from the School of Computer Science at Carnegie Mellon University, and his B.Tech. from the Indian Institute of Technology, Madras. He is interested in developing geometric, optimization, and machine learning algorithms for robotics and automation. Dr. Akella's research interests are in robot motion and manipulation planning, multi-robot coordination, coverage planning, and informative path planning. He has received the NSF CAREER award, and the CCI Excellence in Graduate Teaching Award.