2020 IEEE International Conference on Robotics and Automation (ICRA)
31 May - 31 August, 2020. Paris, France

Line Coverage with Multiple Robots

Saurav Agarwal and Srinivas Akella

***Abstract*— The *line coverage problem* is the coverage of linear environment features (e.g., road networks, power lines), modeled as 1D segments, by one or more robots while respecting resource constraints (e.g., battery capacity, flight time) for each of the robots. The robots incur direction dependent costs and resource demands as they traverse the edges. We treat the line coverage problem as an optimization problem, with the total cost of the tours as the objective, by formulating it as a mixed integer linear program (MILP). The line coverage problem is NP-hard and hence we develop a heuristic algorithm, Merge-Embed-Merge (MEM). We compare it against the optimal MILP approach and a baseline heuristic algorithm, Extended Path Scanning. We show the MEM algorithm is fast and suitable for real-time applications. To tackle large-scale problems, our approach performs graph simplification and graph partitioning, followed by robot tour generation for each of the partitioned subgraphs. We demonstrate our approach on a large graph with 4,658 edges and 4,504 vertices that represents an urban region of about 16 sq. km. We compare the performance of the algorithms on several small road networks and experimentally demonstrate the approach using UAVs on the UNC Charlotte campus road network.**

I. Introduction

Line coverage is the coverage of linear environment features, for tasks such as inspection and monitoring. Linear features include road networks, power lines, and oil and gas lines. The linear features are modeled as 1D segments, and all points along the segments must be visited. Since the features are distributed over large areas, this necessitates the use of multiple robots simultaneously to complete the coverage tasks in a timely manner. Linear infrastructure coverage tasks can be efficiently performed by unmanned aerial vehicles (UAVs) as they can cover linear features over large regions even if the terrain is untraversable by ground robots. This is especially important for rapid assessment and response after natural disasters such as storms and earthquakes.

In a *coverage* application, the robots are required to visit specified features in the environment. Such features may be a set of 2D regions, 1D line features, or points. *Area coverage*, the coverage of 2D regions, has been widely studied [1], [2]. *Point coverage*, the coverage of point features, involves solving node routing problems, such as the traveling salesperson problem and the vehicle routing problem [3]. *Line coverage*, modeled as coverage of all the required edges of an underlying graph, is related to arc routing problems [4]. This problem has not received much attention in the robotics community and is the primary focus of this paper.

*This work was supported in part by NSF Awards IIS-1547175 and IIP-1439695, a UNC IPG award, and DARPA HR00111820055.

The authors are with the Department of Computer Science, University of North Carolina at Charlotte, NC 28223, USA.
sagarw10@uncc.edu, sakella@uncc.edu.

The line coverage problem is modeled using a graph. The edges are classified as *required* and *non-required*. Required edges correspond to the linear features to be covered, and the non-required edges can be used to travel from one vertex to another. The vertices represent the end points of the edges.

There are two modes of travel for the robots. A robot is said to be *servicing* a required edge if task-specific actions such as collecting sensor data are performed. Each required edge is serviced exactly once by one of the robots. A robot may travel from one vertex to another without performing servicing. This is known as *deadheading* and both types of edges may be used any number of times for this purpose. There is a service cost and a deadheading cost (e.g., travel time) associated with each edge and they are incurred each time an edge is serviced or deadheaded, respectively. The sum of the service costs and the deadheading costs of the line coverage problem is to be minimized. As the task-specific sensors are not used during deadheading, the associated deadheading costs are usually lower than the service costs.

Another key aspect of the line coverage problem is modeling of resource constraints (e.g., battery life) in the form of *demands*. Since both servicing and deadheading consume resources, they have associated demands. The amount of total demand a robot can fulfill is constrained by its specified *capacity*. A *depot* or a base location is also specified from where the robots can be launched, recovered, and recharged.

We now define the *line coverage problem* as an optimization problem that finds a set of tours, starting and ending at the depot, that minimizes the total cost of travel, while respecting the capacity constraints of the robots. Figure 1 shows a road network instance of the line coverage problem, the generated tours, and an orthomosaic generated from the UAV images.

In many robotics applications, the cost of travel is direction dependent. For example, for ground robots, the cost of traveling uphill can be significantly higher than that of traveling downhill. Similarly, for UAVs, the cost of an edge may differ in the two directions due to wind conditions. Hence, we consider the graph to have asymmetric edge costs for both servicing and deadheading. Similarly, the service and deadheading demands can also be dependent on the direction of travel, and thus are considered to be asymmetric. For example, ground robots consume more energy going uphill than going downhill, and UAVs flying against the wind consume more energy than when flying with the wind. We use the phrase *fully asymmetric* to emphasize that all four quantities—the two costs and the two demands—are considered to be asymmetric to distinguish the line coverage problem from its special cases.

978-1-7281-7395-5/20/$31.00 ©2020 IEEE

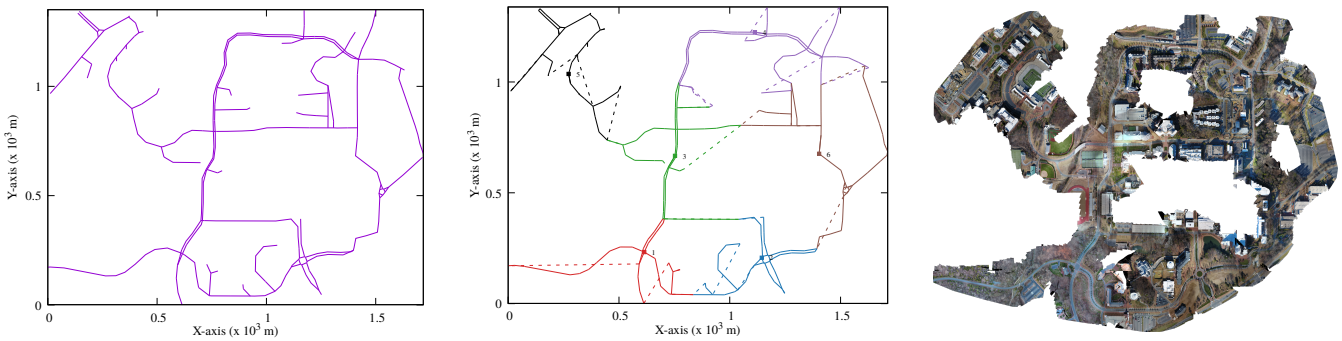

Fig. 1. Line coverage of a road network graph by multiple UAVs. (left) A map of the UNC Charlotte campus. (middle) The graph is clustered into smaller components, each represented by a different color. The generated tours for UAVs covering the graph are shown, with dashed segments indicating deadheading travel. (right) An orthomosaic of the road network computed from photos taken by the UAVs flown autonomously along the generated tours.

The line coverage problem is a generalization of the Capacitated Arc Routing Problem (CARP) [4], described in Section II. The NP-hardness of CARP implies the line coverage problem is NP-hard. This makes it imperative to study heuristic and approximation algorithms.

In this paper we elucidate the line coverage problem using multiple robots with resource constraints, and make the following contributions. (1) We pose the line coverage problem as an optimization problem, and develop a mixed integer linear programming (MILP) model. (2) We develop two heuristic algorithms, Merge-Embed-Merge and Extended Path Scanning, that are computationally efficient. Our algorithms are designed to handle the fully asymmetric nature of the problem—the service and deadheading costs, and service and deadheading demands are all considered to be asymmetric. (3) We present a clustering based approach to efficiently partition large graphs where it may not be possible to reach all the required edges from a single depot location due to resource constraints. We present simulation results on six different real-world road networks, and an experimental validation on the UNC Charlotte campus.

The practical benefits of our line coverage approach are: (1) The algorithms ensure that the required edges are covered efficiently, thereby optimizing the total cost of tours in terms of operation time. (2) In contrast to area coverage, only the relevant features are covered, thus reducing the coverage time, the amount of sensor data, and the time for analysis. (3) The formulation can handle multiple robots, battery life constraints, and asymmetric tour costs and demands.

II. Related Work

Area Coverage: Most work in robotic coverage focuses on area coverage of environments [1], [2]. This includes work on multiple-robot area coverage [5], [6], [7], and some recent work that considers robot battery constraints [8], [9], [10].

Line Coverage: There has been a little work on using UAVs for line coverage tasks. Dille and Singh [11] presented algorithms for a single UAV, with kinematic constraints, to perform coverage of a road network. Oh et al. [12] presented an algorithm for road coverage by a team of UAVs while considering their turn constraints. UAV energy constraints are modeled using a knapsack formulation.

Arc Routing Problems: Arc Routing Problems (ARP) consider the task of covering a specified set of edges of a graph by one or more vehicles [4]. The ARP has not been used much in robotics, in contrast to the Vehicle Routing Problem (VRP) [13], where the tours must visit nodes of the graph. The ARP is usually applied to transportation problems in which servicing is related to tasks such as delivery and pickup of goods. Hence the travel times are used as costs, and have the same value whether the edge is serviced or deadheaded. Separate service and deadheading costs are typically not considered, and are usually symmetric even when considered.

In a Rural Postman Problem (RPP), the vehicle capacities are not considered. Easton and Burdick [14] introduced kRPP, RPP with k vehicles. They modeled coverage of 2D object boundaries as a kRPP and presented a cluster first and route second heuristic. Xu and Stentz [15] considered the kRPP for street coverage with incomplete prior map information.

The Capacitated Arc Routing Problem (CARP) [16], the most relevant variant, is a special case of the line coverage problem—there are no deadheading demands, and all costs and demands are symmetric. Since the CARP is NP-hard [16], several heuristic algorithms [4] have been developed for the CARP. An approximation algorithm A-ALG, with an approximation factor of $7/2$, was presented in [17].

Several variants of the ARP partially and individually address a subset of the features of the line coverage problem (Table I). The Windy Rural Postman problem (WRPP) [18] considers asymmetric travel costs. The CARP with deadheading demands (CARP-DD) was introduced by Sipahioglu et al. [19], and studied further in [20], [21]. The CARP-DD does not consider asymmetric graphs. Gouveia et al. [22] presented a lower bound approach for the CARP on mixed graphs using mixed integer linear programming. Its exponential complexity makes it unsuitable for large scale robotics applications. All these variants are NP-hard. Since the line coverage problem generalizes all these CARP variants, it poses significant computational challenges.

TABLE I

THE LINE COVERAGE PROBLEM, WITH ITS SPECIAL CASES

Literature	Number of robots	Deadheading		Service	
		Cost	Demand	Cost	Demand
Line coverage	K	A	A	A	A
Mixed-CARP [22]	K	A	–	A	A
CARP-DD [19]	K	S	S	–*	S
CARP [4]	K	S	–	–*	S
WRPP [18]	1	A	–	–*	–
kRPP [14], [15]	K	S	–	–*	–

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

III. LINE COVERAGE PROBLEM

We now define the *line coverage problem*. We are given a connected graph $G = (V, E, E_R)$ where V is the set of vertices, E is the set of edges, and $E_R \subseteq E$ is the set of required edges. The task is to find a set of tours on the graph that minimizes the total cost of travel, such that (1) the tours start and end at a specified depot location $v_0 \in V$, (2) all the required edges are serviced, and (3) none of the tours exceed the capacity Q of the robots. The costs and the demands are provided as inputs along with the graph.

Each edge is represented by a pair of vertices (i, j). When a robot traverses edge (i, j) from vertex i to vertex j, we also say *directed edge* (i, j) to emphasize the direction of travel.

We now describe the costs and demands. (1) If a robot *services* a required edge $(i, j) \in E_R$ from vertex i to vertex j, then a service cost c^s_{ij} and a service demand q^s_{ij} are incurred. These are also denoted as $c^s(i, j)$ and $q^s(i, j)$. (2) If a robot traverses an edge without servicing it, the robot is said to be *deadheading* and deadheading cost and demand are incurred. This occurs when a robot is traveling from a vertex of a required edge to that of another required edge, and both required and non-required edges may be deadheaded. Deadheading cost is denoted by c^d_{ij} and demand by q^d_{ij} for edge $(i, j) \in E$ from vertex i to vertex j. These are also denoted as $c^d(i, j)$ and $q^d(i, j)$. When the triangle inequality does not hold for a graph, or a direct path does not exist between two given vertices i and j, c^d_{ij} and q^d_{ij} correspond to the shortest cost path from i to j.

We consider the edge costs and demands, for both service and deadheading, to be direction dependent (for example, $c^s_{ij}(q^s_{ij})$ may differ from $c^s_{ji}(q^s_{ji})$). The service and deadheading costs and demands can all be arbitrary positive numbers. The costs affect the objective function, and the demands are related to the robot capacity, i.e., the sum of demands for a tour should not exceed the robot capacity Q. For example, the travel time may constitute the costs, and the energy consumed may represent the demands.

The line coverage problem can be thought of as a *fully asymmetric CARP with deadheading demands*, i.e., the service and deadheading costs, and service and deadheading demands are all asymmetric. The fully asymmetric nature of the problem, coupled with deadheading demands, poses serious computational challenges, especially when applications require fast generation of tours on large graphs.

A. Mixed Integer Linear Programming Formulation

We develop a mixed integer linear programming (MILP) formulation for the line coverage problem, by extending that for CARP on mixed graphs [22] and incorporating deadheading demands. In a mixed graph, arcs are used to represent one-way streets. Hence we assign two arcs (representing the two directions of travel) for each of the edges in the original graph. The formulation uses network flows to represent the demands. The depot v_0 (i.e., vertex 0) acts as the source for the flow and the flow is absorbed along the arcs. The flow constraints ensure that the generated tours are connected and the robot capacities are respected.

MILP formulation and notation:

- K: Maximum number of tours; to be specified
- A: Set of arcs $\{(i, j), (j, i) : (i, j) \in E\}$
- A_R: $\{(i, j), (j, i) : (i, j) \in E_R\}$, $A_R \subseteq A$
- x^k_{ij}: Binary variable denoting whether arc $(i, j) \in A_R$ is serviced in tour k
- y^k_{ij}: Integer variable denoting the number of times arc $(i, j) \in A$ is deadheaded in tour k
- f^k_{ij}: Continuous variable denoting the flow across arc $(i, j) \in A$ in tour k
- Q, λ: Robot capacity, and tour setup cost

Minimize:

$$\sum_{k=1}^{K} \left[\sum_{(i,j)\in A_R} c^s_{ij} x^k_{ij} + \sum_{(i,j)\in A} c^d_{ij} y^k_{ij} + \lambda \sum_{(0,i)\in A_R} x^k_{0i} + \lambda \sum_{(0,i)\in A} y^k_{0i} \right] \tag{1}$$

subject to:

$$\sum_{j:(i,j)\in A_R} x^k_{ij} + \sum_{j:(i,j)\in A} y^k_{ij} = \sum_{j:(j,i)\in A_R} x^k_{ji} + \sum_{j:(j,i)\in A} y^k_{ji} \quad \forall i, k \tag{2}$$

$$\sum_{k=1}^{K} (x^k_{ij} + x^k_{ji}) = 1 \quad \forall (i, j) \in E_R \tag{3}$$

$$\sum_{j:(0,j)\in A_R} x^k_{0j} + \sum_{j:(0,j)\in A} y^k_{0j} \leq 1 \quad \forall k \tag{4}$$

$$\sum_{j:(j,i)\in A} f^k_{ji} - \sum_{j:(i,j)\in A} f^k_{ij} = \sum_{j:(j,i)\in A_R} q^s_{ji} x^k_{ji} + \sum_{j:(j,i)\in A} q^d_{ji} y^k_{ji} \quad \forall k,\ i \neq 0 \tag{5}$$

$$\sum_{j:(0,j)\in A} f^k_{0j} = \sum_{(i,j)\in A_R} q^s_{ij} x^k_{ij} + \sum_{(i,j)\in A} q^d_{ij} y^k_{ij} \quad \forall k \tag{6}$$

$$\sum_{i:(i,0)\in A} f^k_{i0} = \sum_{i:(i,0)\in A_R} q^s_{i0} x^k_{i0} + \sum_{i:(i,0)\in A} q^d_{i0} y^k_{i0} \quad \forall k \tag{7}$$

$$f^k_{ij} \leq Q(x^k_{ij} + y^k_{ij}) \quad \forall (i, j) \in A, \forall k \tag{8}$$

$$x^k_{ij} \in \{0, 1\} \quad \forall (i, j) \in A_R,\ \forall k \tag{9}$$

$$f^k_{ij} \geq 0,\ y^k_{ij} \geq 0, y^k_{ij} \in \mathbb{Z} \quad \forall (i, j) \in A,\ \forall k \tag{10}$$

It is assumed that the robot capacity Q is sufficiently large to service any of the edges, i.e., $Q > \min\left(q^d_{0i} + q^s_{ij} + q^d_{j0}, q^d_{0j} + q^s_{ji} + q^d_{i0}\right)\ \forall (i, j) \in E_R$.

The objective function (1) is the total cost of the mission and is to be minimized. The constraints (2) ensure connectivity of tours at each vertex—the number of traversed

arcs exiting a vertex is equal to number of traversed arcs entering the vertex. Constraints (3) ensure all the required edges are serviced. Constraints (5) are the generalized flow constraints at vertex i such that $q^s_{ji} + q^d_{ji}$ units of flow are absorbed if tour k services (and/or deadheads) edge (j, i) in the direction $j \to i$. Constraints (6) and (7) specify the amount of flow coming out of and into the depot vertex, respectively. Constraints (5)–(8) ensure that the tours are connected. Constraints (8), along with (4) and (6), guarantee that the robot capacities are respected. Constraints (5)–(8) all include modifications to incorporate deadheading demands. Additional constraints that give tighter bounds for the MILP formulation and eliminate some of the symmetric solutions are also used. We have extended the proofs in [22] for tour capacity constraints and connectivity. (The additional constraints and proofs are omitted for space reasons.)

The MILP formulation gives the optimal solution if such a solution exists. Since the problem is NP-hard, we next present heuristic algorithms to solve it on large maps.

IV. Heuristic Line Coverage Algorithms

The heuristic algorithms in the literature for the CARP [4], [23] are not directly applicable to the line coverage problem, due to the asymmetric costs and demands and inclusion of deadheading demands. In this section, we first present our extension of the Path Scanning heuristic to the line coverage problem. We then present a new heuristic algorithm, Merge-Embed-Merge. In both the heuristic algorithms, the number of tours is computed by the algorithm.

A. Extended Path Scanning: A Greedy Approach

Path scanning is a greedy heuristic algorithm originally proposed for the *Chinese Postman Problem* (CPP) [24]. The original algorithm does not consider deadheading demands and was not designed for fully asymmetric graphs. We extended the heuristic algorithm to the line coverage problem, and use it as a simple and fast heuristic for obtaining a baseline solution. The algorithm builds tours starting from the depot by adding the closest (unserviced) required edges. For each required edge two arcs, corresponding to the two directions, are maintained in a list $\mathcal{F}$. Initially, $\mathcal{F} = A_R$. Before a new arc is appended to a tour, the algorithm ensures that the robot has sufficient residual capacity to return to the depot via the shortest path. This is done by checking if the sum of (a) current total demand for the tour, (b) the demand for the new arc, and (c) the demand to return back to the depot, is less than the capacity of the robot. For a tour r with l as the vertex at the end of the current sequence of required edges, the constraint for checking if a required directed edge (i, j) can be added is: $\text{demand}(r) + q^d(l, i) + q^s(i, j) + q^d(j, v_0) \leq Q$. The cost of the tour is then updated to be $\text{cost}(r) \leftarrow \text{cost}(r) + c^d(l, \bar{i}) + c^s(\bar{i}, \bar{j})$, where $(\bar{i}, \bar{j})$ corresponds to the arc with least deadheading cost: $(\bar{i}, \bar{j}) = \text{argmin}\{\big(c^d(l, i)\big) : (i, j) \in \mathcal{F}\}$. The arc $(\bar{i}, \bar{j})$, and its opposite arc $(\bar{j}, \bar{i})$, are both then removed from $\mathcal{F}$ after being appended to a tour.

The end of the sequence of required edges is then updated to $l \leftarrow \bar{j}$. If no such arc can be added, the tour returns to the depot and ends, and the final cost is given by $\text{cost}(r) \leftarrow \text{cost}(r) + c^d(l, v_0) + \lambda$, where λ is the setup time. The algorithm generates new tours until all the required edges belong to one of the tours.

When there are equidistant arcs, tie-breaking rules can be used [4]. We found that selecting an equidistant arc at random [25], and running the entire algorithm several times, produces the best results. The number of times we run the algorithm is proportional to the number of vertices with more than two incident edges. The set of tours with the least total cost over all the runs is returned. The running time of the Extended Path Scanning (EPS) algorithm is $\mathcal{O}(M^2)$ for each run, where M is the number of required edges.

B. Merge-Embed-Merge: A Greedy Constructive Heuristic

We present a new algorithm, Merge-Embed-Merge (MEM), for the line coverage problem. The underlying concept of the algorithm is to start with M tours, one per required edge. Subsequently, the tours are merged together in greedy fashion to form a smaller set of tours. This concept was first presented in the *Clarke and Wright Heuristic* [26] for the capacitated vehicle routing problem, and later adopted in the *Augment-Merge algorithm* [24] for CARP. However, the Augment-Merge algorithm cannot handle the asymmetric costs and demands, as well as the deadheading demands of the line coverage problem.

The Merge-Embed-Merge algorithm, given in Algorithm 1, is composed of four components: (1) *initialization* of tours, (2) computation of *savings*, (3) *merging*, and (4) *embedding* the new merged tour.

Representation of tours: A tour is represented by a sequence of required edges to be serviced by the robot performing the tour. Starting from the depot, the robot travels to the starting vertex of the first required edge, services the sequence of required edges, and returns to the depot. Let a tour R_p contain vertices i and j such that i is the starting vertex of the first required edge, and vertex j is the ending vertex of the last required edge serviced by the tour. The robot will deadhead from the depot v_0 to vertex i, serve required edges starting from i up to the last required edge ending at vertex j, and then deadhead back to v_0. Note that the paths $v_0 \to i$ and $j \to v_0$ are shortest paths and can include several edges, all being deadheaded. If two successive required edges e_s and e_t are not adjacent, the robot deadheads along the shortest path from the ending vertex of e_s to the starting vertex of e_t. Thus, the sequence $i \to j$ itself may involve deadheading between the containing required edges. Then the cost of the tour R_p is $\text{cost}(p) = c(v_0, i) + c(i, j) + c(j, v_0) + \lambda$, where λ is the tour setup cost.

Initialization of tours: Initially a tour is built for each of the required edges, comprised of the shortest path $v_0 \to i$ from the depot v_0 to one of the vertices i of the edge, the edge (i, j) itself, and then from the other vertex j of the edge back to the depot $j \to v_0$. As the edges have asymmetric costs the tour in the direction with the lower cost is used.

Algorithm 1: Merge-Embed-Merge algorithm

Input : $G(V, E, E_R)$, costs, demands, capacity Q
Output: Array R of tours, where each tour R_i is a sequence of required edges

```
/* Initialization of tours                                   */
r ← 1;
for (i, j) ∈ E_R do
    c ← c^d(v_0, i) + c^s(i, j) + c^d(j, v_0) + λ;
    d ← q^d(v_0, i) + q^s(i, j) + q^d(j, v_0);
    c̄ ← c^d(v_0, j) + c^s(j, i) + c^d(i, v_0) + λ;
    d̄ ← q^d(v_0, j) + q^s(j, i) + q^d(i, v_0);
    cost(r) ← min(c, c̄);
    R_r ← if c ≤ c̄ then (i, j), else (j, i);
    demand(r) ← if c ≤ c̄ then d, else d̄;
    r ← r + 1;
/* Compute savings                                           */
S ← ∅;
foreach pair of tours R_p, R_q do
    Compute savings s_pq for R_p ⊎ R_q;
    if s_pq ≥ 0 and demand(R_p ⊎ R_q) ≤ Q then
        S.push((p, q, s_pq));
make_heap(S); /* max-heap                                   */
/* Repeated Merge and Embed                                  */
while S ≠ ∅ do
    (p, q, s) ← S.extract-max();
    if R_p ≠ ∅ and R_q ≠ ∅ then
        /* Merge                                             */
        R_r ← R_p ⊎ R_q; r ← r + 1;
        R_p ← ∅; R_q ← ∅;
        /* Embed                                             */
        foreach tour R_i such that i ≠ r and R_i ≠ ∅ do
            Compute savings s_ri for R_r ⊎ R_i;
            if s_ri ≥ 0 and demand(R_r ⊎ R_i) ≤ Q then
                S.insert((r, i, s_ri));
```

The tours are added to an array R, and the initial number of tours is equal to the number of required edges M.

Compute savings: Consider two tours, R_p with sequence $i \to j$, and R_q with sequence $m \to l$, as potential candidates for merging. There are eight possible permutations to merge the two tours, of which four are shown in Fig. 2. The remaining four ways consist of tours in the reverse directions. Consider the first merge sequence, $v_0 \to i \to j \to m \to l \to v_0$. The cost of this merged tour, $R(pq)$, is $\text{cost}(pq) = c(v_0, i) + c(i, j) + c(j, m) + c(m, l) + c(l, v_0) + \lambda$. There is a saving in costs since we no longer need $j \to v_0$ and $v_0 \to m$, and there is also a saving of a tour setup cost. Thus, the net savings s_{pq}, for merging tours R_p and R_q, is given by $\text{cost}(p) + \text{cost}(q) - \text{cost}(pq)$. However, as the costs are asymmetric, any change in directions of the segments will affect the savings. Some of the eight permutations might not satisfy the capacity constraint Q of the robots. We denote by $R_p \uplus R_q$ the merged tour that has the maximum savings and satisfies the capacity constraint. If no such combination exists, then $R_p \uplus R_q = \emptyset$ and saving $s_{pq} = -\infty$.

The maximum savings for merging each pair of tours (that yield a feasible tour) is computed, forming a tuple (p, q, s_{pq}), where p and q are the indices of the tours considered and s_{pq} is the corresponding savings. These $M(M-1)/2$ tuples are stored in a binary max-heap S, built in $\mathcal{O}(M^2)$ time.

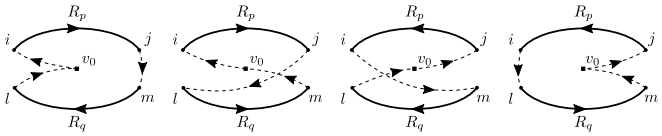

Fig. 2. Four of the eight different permutations to merge two tours. The remaining four permutations consist of the shown permutations in the reverse directions. The first merged tour is $v_0 \to i \to j \to m \to l \to v_0$, and its reverse direction tour is $v_0 \to l \to m \to j \to i \to v_0$.

Next the merge and embed steps are executed repeatedly until no further merges are possible.

Merge: The maximum element from the max-heap S is extracted and the constituent tours are merged (if neither is empty), i.e., the tours are merged *greedily* to maximize the immediate savings. The cost of the newly merged tour R_r is computed, and the corresponding constituent tours are set to $\emptyset$. The merged tour R_r is appended to the array R. This step has complexity $\mathcal{O}(\log |S|)$, where $|S|$ is the number of elements in S.

Embed: Now we embed the savings due to the newly merged tour R_r into the max-heap S so that it is considered for future merges. Potential savings are computed for the new tour R_r if merged with each of the other nonempty tours in the array R. New tuples (r, i, s_{ri}) are generated and inserted into the max-heap S. As there are $|R| - 1$ such new tuples, this component has a complexity of $\mathcal{O}(|R| \log(|S| + |R|))$.

The merge and embed components are executed until no further merges are possible, i.e., $S = \emptyset$. The maximum number of tours in the array R is upper bounded by $2M$, with at most M non-empty tours. The maximum number of elements in the max-heap S is $\mathcal{O}(M^2)$, and the complexity of the repeated merge-embed component over all possible merges is $\mathcal{O}(M^2 \log M)$. Thus, the overall complexity of the algorithm is $\mathcal{O}(M^2 \log M)$.

V. Integrated Approach for Line Coverage

We now discuss our overall approach for solving the line coverage problem for large maps. First, a graph simplification step based on the field of view (FoV) of the sensor and the flight altitude of the UAV is outlined. We then describe the clustering approach used to partition the graph and compute the depot locations. The MILP and heuristic algorithms are then used to generate tours for each of the subgraphs.

A. Graph Simplification

The numbers of vertices and required edges can be quite large because the curves in the (road) network are modeled as polylines. However, owing to the sensor FoV and the high altitude of the UAVs, we can simplify the graph. An algorithm, based on depth-first search, simplifies the graph while ensuring UAV coverage of the simplified graph will still cover the original graph. For each polyline discovered in the algorithm, we attempt to connect its vertices directly to increasingly distant vertices while ensuring all intermediate vertices are in the swept FoV of a UAV flying along a straight-line path. This provides a simpler polyline with fewer edges and vertices, while ensuring the input polyline is still covered. The complexity of the algorithm is $\mathcal{O}(V + E)$.

TABLE II

PERFORMANCE OF THE ALGORITHMS ON DIFFERENT ROAD NETWORKS

Road network	No. of edges*	Area (km^2)	Network length (km)	No. of clusters	No. of tours	Compute time (s)				Total of tour costs (s)		
						MILP	MILP WS†	MEM	EPS	MILP	MEM	EPS
Mumbai	329 (220)	0.86	13.56	3	5	83	74	0.050	0.174	3304	3416	3608
Rio	407 (264)	1.65	20.00	6	8	16	8	0.038	0.080	4792	4986	5204
Manhattan	544 (440)	2.71	38.33	5	13	7919	3718	0.073	0.321	9049	9446	9801
Naples	644 (232)	0.90	11.20	4	5	66	33	0.042	0.147	2731	2843	3035
UNCC	768 (282)	2.29	14.16	6	6	24	3	0.055	0.067	3483	3573	3897

The cost listed for each method is the corresponding total flight time of all tours, without the setup cost. The MEM solutions are within 2–5% of the optimal MILP solutions, and the EPS solutions are within 8–12% of the optimal MILP solutions. *: The number of edges in the simplified graphs are in parentheses. †: MILP with warm start using MEM solution. The dataset is available at: https://github.com/AgarwalSaurav/LineCoverage-dataset.

B. Graph Partitioning and Depot Location

The MILP formulation and heuristic algorithms assume a single depot. In large networks, the demand required to service an edge may be greater than the robot capacity because of the potentially large distances to the depot. Hence we use k-medoids clustering to cluster the edges, with a distance function that computes the minimum of all eight possible pair-wise travel costs between the vertices of two edges. As we are clustering the edges, the medoids are themselves edges and one can choose either of the two vertices of the medoid edge as a depot location. Edge clustering partitions the graph into smaller subgraphs, which enables generation of feasible tours. Furthermore, having multiple depots decreases the deadheading travel of the robots. Additionally, tours in multiple clusters can be executed simultaneously. Each robot starts and ends its tours at its corresponding depot, and services only the required edges in the corresponding subgraph.

VI. LINE COVERAGE RESULTS

A. Small Scale Examples and Experimental Results

We selected five representative small road networks using OpenStreetMap [27], and compared the performance of the MILP and heuristic algorithms on them (Table II). We also used the solutions generated by the MEM heuristic algorithm to determine (an upper bound on) the number of tours in the MILP formulation, and also to warm start the MILP. Warm starts significantly reduce the time taken for the MILP.

The heuristic algorithms were implemented in C++, and the MILP was solved using Gurobi [28]. They were executed on a 2.60 GHz Intel i9 desktop. The MEM and EPS heuristic algorithms generate solutions within 2–5% and 8–12% of the optimal solution. The MEM heuristic took less than 0.1 s for each map, and consistently outperformed the EPS heuristic both in computation time and the quality of the solutions.

The service and deadheading costs were calculated assuming a UAV deadheading speed of 10 $\mathrm{m\,s}^{-1}$ and a servicing speed of 5 $\mathrm{m\,s}^{-1}$. The service and deadheading demands were set proportional to the service and deadheading costs, respectively. The maximum tour capacity (i.e., battery limit) was set to a conservative flight time of 900 s. A simulated wind of speed 2 $\mathrm{m\,s}^{-1}$ at an angle of $\frac{\pi}{4}$ rad to the X-axis from the southwest was assumed.

For the UNC Charlotte campus map, covering an area of 2.29 km^2, we experimentally validated the generated routes using two DJI Phantom 4 UAVs flown autonomously. Graph simplification reduced the map to 263 vertices and 282 required edges from the original 749 vertices and 768 required edges. Fig. 1 shows the map, the generated clusters and tours, and the orthomosaic computed from the tours.

B. Large Scale Example

We performed road coverage, in simulation, of a region around UNC Charlotte approximately 16 km^2 in area. The input graph representing the road network consists of 4,504 vertices and 4,658 required edges. Graph simplification reduces the graph size to 1,872 vertices and 2,018 required edges. There are about 4 million non-required edges representing a complete graph as the UAVs can travel from any vertex to any other. The number of non-required edges do not directly affect the running time of the heuristic algorithms since the shortest path (by direct flight) from one vertex to another is precomputed. With 10 clusters, the MILP formulation did not converge after 5 days of computation. The EPS heuristic took 3.8 s, while MEM was able to obtain the solutions in 0.6 s for all the clusters.

VII. CONCLUSION

We have presented an approach for the coverage of linear environment features by a set of robots, given energy constraints, and costs and demands that are asymmetric for both service and deadheading. We developed an MILP formulation and two efficient heuristic algorithms for the line coverage problem. To handle line coverage over large regions, we performed simplification of the network graph, partitioned it into subgraphs, and generated tours for each subgraph by solving the corresponding line coverage problem. We presented our algorithms on six different real-world road networks, and demonstrated that the MEM algorithm is efficient and fast for robotics applications. We also used tours generated by the algorithms to autonomously fly UAVs and generate an orthomosaic of the UNC Charlotte campus road network.

An alternative approach we are now exploring is to initially generate a single tour assuming infinite robot capacity, and then apply a tour-splitting algorithm to generate smaller tours that respect the robot capacity. Future work includes extending this approach to develop polynomial-time algorithms with approximation guarantees. We also plan to extend our work to incorporate turning costs and nonholonomic constraints into these algorithms.

REFERENCES

[1] H. Choset, "Coverage for robotics – A survey of recent results," *Annals of Mathematics and Artificial Intelligence*, vol. 31, no. 1, pp. 113–126, Oct. 2001.

[2] E. Galceran and M. Carreras, "A survey on coverage path planning for robotics," *Robotics and Autonomous Systems*, vol. 61, no. 12, pp. 1258–1276, Dec. 2013.

[3] D. G. Macharet and M. F. M. Campos, "A survey on routing problems and robotic systems," *Robotica*, vol. 36, no. 12, pp. 1781–1803, Dec. 2018.

[4] A. Corberan and G. Laporte, Eds., *Arc Routing: Problems, Methods, and Applications*. Philadelphia, PA: Society for Industrial and Applied Mathematics (SIAM), 2014.

[5] I. Rekleitis, A. P. New, E. S. Rankin, and H. Choset, "Efficient Boustrophedon multi-robot coverage: an algorithmic approach," *Annals of Mathematics and Artificial Intelligence*, vol. 52, no. 2, pp. 109–142, 2008.

[6] I. Maza and A. Ollero, "Multiple UAV cooperative searching operation using polygon area decomposition and efficient coverage algorithms," in *Distributed Autonomous Robotic Systems 6*. Tokyo: Springer Japan, 2007, pp. 221–230.

[7] G. S. C. Avellar, G. A. S. Pereira, L. C. A. Pimenta, and P. Iscold, "Multi-UAV routing for area coverage and remote sensing with minimum time," *Sensors*, vol. 15, no. 11, pp. 27 783–27 803, 2015.

[8] G. P. Strimel and M. M. Veloso, "Coverage planning with finite resources," in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, Chicago, IL, USA, Sep. 2014, pp. 2950–2956.

[9] I. Shnaps and E. D. Rimon, "Online coverage of planar environments by a battery powered autonomous mobile robot," *IEEE Transactions on Automation Science and Engineering*, vol. 13, no. 2, pp. 425–436, 2016.

[10] K. Yu, J. M. O'Kane, and P. Tokekar, "Coverage of an environment using energy-constrained unmanned aerial vehicles," in *IEEE International Conference on Robotics and Automation*, Montreal, Canada, May 2019, pp. 3259–3265.

[11] 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.

[12] 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.

[13] P. Toth and D. Vigo, Eds., *Vehicle Routing: Problems, Methods, and Applications*, 2nd ed. Philadelphia, PA: Society for Industrial and Applied Mathematics (SIAM), 2014.

[14] 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.

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

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

[17] 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.

[18] E. Benavent, A. Corberan, E. Pinana, I. Plana, and J. M. Sanchis, "New heuristic algorithms for the windy rural postman problem," *Computers & Operations Research*, vol. 32, no. 12, pp. 3111–3128, 2005.

[19] A. Sipahioglu, G. Kirlik, O. Parlaktuna, and A. Yazici, "Energy constrained multi-robot sensor-based coverage path planning using capacitated arc routing approach," *Robotics and Autonomous Systems*, vol. 58, no. 5, pp. 529–538, 2010.

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

[21] E. Bartolini, J.-F. Cordeau, and G. Laporte, "An exact algorithm for the capacitated arc routing problem with deadheading demand," *Operations Research*, vol. 61, no. 2, pp. 315–327, 2013.

[22] 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.

[23] S. Wøhlk, "A decade of capacitated arc routing," in *The Vehicle Routing Problem: Latest Advances and New Challenges*. Boston, MA: Springer US, 2008, pp. 29–48.

[24] 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.

[25] J. M. Belenguer, E. Benavent, P. Lacomme, and C. Prins, "Lower and upper bounds for the mixed capacitated arc routing problem," *Computers & Operations Research*, vol. 33, no. 12, pp. 3363–3383, 2006.

[26] 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.

[27] OpenStreetMap contributors, "Planet dump retrieved from https://planet.osm.org," https://www.openstreetmap.org, 2017.

[28] Gurobi Optimization, LLC, "Gurobi optimizer reference manual," 2020. [Online]. Available: http://www.gurobi.com.