Received: 20 December 2022 | Revised: 6 April 2023 | Accepted: 31 May 2023

DOI: 10.1002/net.22171

SPECIAL ISSUE ARTICLE

WILEY

The single robot line coverage problem: Theory, algorithms, and experiments

Saurav Agarwal | **Srinivas Akella**

Department of Computer Science, University of North Carolina at Charlotte, Charlotte, North Carolina, USA

Correspondence
Saurav Agarwal, Department of Computer Science, University of North Carolina at Charlotte, 9201 University City Blvd, Charlotte, NC 28223, USA.
Email: sagarw10@charlotte.edu

Funding information
Defense Advanced Research Projects Agency, Grant/Award Number: HR00111820055; National Science Foundation, Grant/Award Number: IIP-1919233, IIS-1547175; University of North Carolina: Inter-institutionalPlanningGrant.

Abstract

Line coverage is the task of servicing a given set of one-dimensional features in an environment. It is important for the inspection of linear infrastructure such as road networks, power lines, and oil and gas pipelines. This paper addresses the single robot line coverage problem for aerial and ground robots by modeling it as an optimization problem on a graph. The problem belongs to the broad class of arc routing problems and is closely related to the rural postman problem (RPP) on asymmetric graphs. The paper presents an integer linear programming formulation with proofs of correctness. Using the minimum cost flow problem, we develop approximation algorithms with guarantees on the solution quality. These guarantees also improve the existing results for the asymmetric RPP. The main algorithm partitions the problem into three cases based on the structure of the *required graph*, that is, the graph induced by the features that require servicing. We evaluate our algorithms on road networks from the 50 most populous cities in the world, consisting of up to 730 road segments. The algorithms, augmented with improvement heuristics, run within 3 s and generate solutions that are within 10% of the optimum. We experimentally demonstrate our algorithms with commercial UAVs on the UNC Charlotte campus road network.

KEYWORDS

arc routing problems, inspection with UAVs, line coverage, path planning for mobile robots, postman problems

1 | INTRODUCTION

Line coverage is the task of servicing linear environment features using sensors or tools mounted on a robot. The features to be serviced are modeled as one-dimensional segments (or curves); all points along the segments must be visited. Consider a natural disaster scenario such as flooding in which an uncrewed aerial vehicle (UAV) with cameras is deployed for the assessment of connectivity of a road network for emergency services. The UAV must traverse the line segments corresponding to the road network and use its cameras to capture images. It may also travel at higher speeds from one point to another while not capturing images. The following question then arises: How should a tour for the robot be planned such that it traverses each road network segment and minimizes the flight time? Figure 1 depicts such a scenario with an optimal tour for a UAV and an orthomosaic generated from the images collected during the flight. Power lines and oil and gas pipelines have similar linear infrastructure that requires frequent inspection. Additional applications arise in perimeter inspection and surveillance, traffic analysis of road

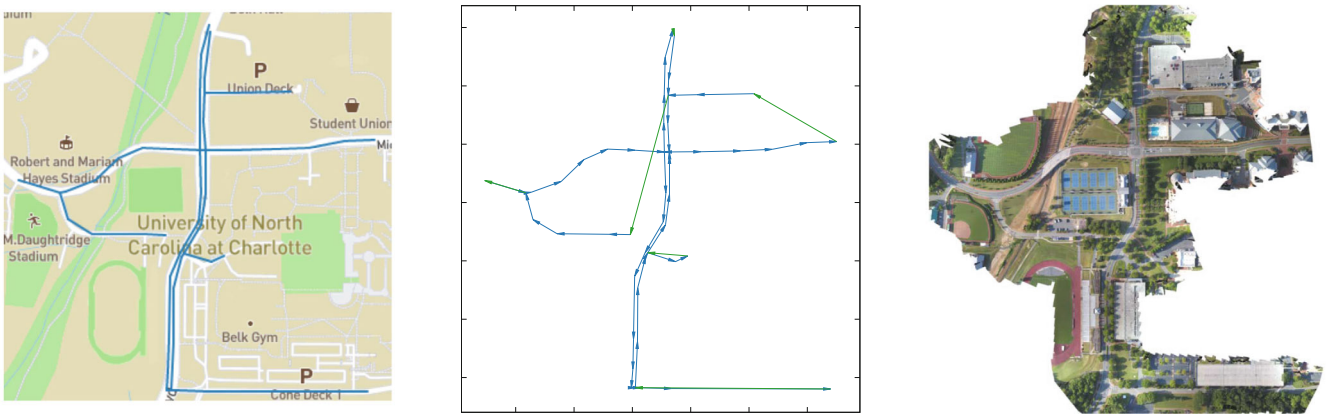


FIGURE 1 Line coverage of a road network by an autonomous UAV: (A) A region of the UNC Charlotte campus road network; blue lines show required edges to be serviced. Non-required edges, not shown, are straight lines between pairs of vertices. The network data was extracted using OpenStreetMap. (B) An optimal coverage tour for the road network is shown; dashed segments indicate deadheading travel. (C) An orthomosaic map of the road network generated from photos taken by the UAV along the required edges of the coverage tour.

networks, and welding operations. Line coverage algorithms can also be used as a subroutine for routing in area coverage problems, that is, coverage of 2D regions, by decomposing the environment into line segments.

Line coverage is closely related to arc routing problems (ARPs) [11]. ARPs have been used for snow plowing, spraying salt, and cleaning road networks [10]. ARPs and their algorithms have been traditionally designed for human-operated vehicles. The above tasks can potentially be automated with uncrewed ground vehicles (UGVs). Recently, ARP variants, such as the drone arc routing problem, have been developed specifically for UAVs. However, line coverage has received limited attention in the robotics community. Developing algorithms that rapidly generate high-quality solutions is essential for planning robot motions. Algorithms with low computation requirements can be deployed conveniently on robots and executed to replan routes if the environment changes. In this paper, we design algorithms for line coverage using autonomous systems such as UAVs and UGVs. The algorithms provide theoretical guarantees on the quality of the solutions, which improve existing results for related single vehicle ARPs. The simulations and experiments further demonstrate the effectiveness of our algorithms for line coverage applications using robots.

The line coverage problem, modeled using a graph, has two defining attributes: (1) The edges in the graph are classified as required and non-required, and (2) Robots have two modes of travel—servicing and deadheading. *Required edges* correspond to the linear features to be covered, and the *non-required edges* can be used by a robot to travel from one vertex to another to reduce cost. The vertices in the graph represent the endpoints of the edges. The 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. The robot may also traverse an edge without performing servicing tasks to optimize the travel time, conserve energy, or reduce the amount of sensor data. This 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 sum of the service and deadhead costs is to be minimized. As task-related actions are performed only when servicing a required edge, not while deadheading, the service costs are considered to be greater than or equal to the deadhead costs. For example, with travel time as the cost, a UAV servicing an edge by recording images may need to travel slower than when deadheading to avoid motion blur. In contrast, the service and deadhead costs are usually assumed to be identical for the required edges in standard ARPs, and therefore, the line coverage problem generalizes the standard problems.

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 along the two directions due to wind conditions. Hence, we consider the graph to have asymmetric edge costs for both servicing and deadheading. Asymmetric edges can also model one-way streets for ground robots.

The *single robot line coverage problem* is the problem of finding a coverage tour that minimizes the total travel cost while ensuring that each linear feature is serviced exactly once. The formulation allows distinct costs for servicing and deadheading and permits asymmetric costs. The single robot line coverage problem is a generalization of the rural postman problem (RPP), introduced by Orloff [33]. The NP-hardness of the RPP, shown by Lenstra and Kan [27], implies that the single robot line coverage problem is NP-hard. This makes it imperative to develop approximation algorithms. Agarwal and Akella [1] addressed the line coverage problem with multiple robots and presented two heuristic algorithms. These heuristic algorithms do not have

theoretical guarantees, though they were empirically shown to give results close to optimal. The single robot line coverage problem, as discussed in this paper, is a special case of the multiple robot version that does not consider the robot resource capacity and the resource demands of edges. Algorithms for the single robot problem can be used to develop algorithms for multiple robots.

Contributions: In this paper, we elucidate the single robot line coverage problem and develop approximation algorithms. We analyze the problem in stages—going from a simpler problem to the most general version. The problems are based on the structure of the *required graph*, that is, the graph induced by only the required edges. The contributions of the paper are:

1. We pose the single robot line coverage problem as an optimization problem and develop an integer linear programming (ILP) formulation that gives an optimal solution. Additionally, we provide formal proofs for the correctness of the formulation.
2. We develop a linear relaxation of the ILP formulation and model it using a minimum cost flow problem. The model is used to design an optimal algorithm for graphs with Eulerian required graphs. A 2-approximation algorithm is then developed for connected required graphs.
3. An $(\alpha(C) + 2)$-approximation algorithm is presented for the general case of a required graph with multiple connected components, where C is the number of connected components, and $\alpha(C)$ is the approximation factor for an algorithm for the asymmetric traveling salesperson problem. Proofs for the approximation guarantee are provided for all the algorithms.
4. We publish a dataset[1] consisting of road networks of the 50 most populous cities in the world and perform simulations. The simulations showcase the modeling of cost functions for robotics applications. The results show that the algorithms compute solutions within 10% of the optimum. The algorithms find solutions to the instances within 3 s and are fast enough to enable rapid robot replanning. Experimental validation of the algorithms is performed. We also provide an open-source implementation[2] of our algorithms.

Building on the earlier publication by the authors [2], this paper develops a thorough theoretical analysis of the formulations and the algorithms, provides extensive simulation results, and validates the algorithms in experiments with UAVs. In particular, formal proofs for the correctness of the ILP formulation are provided. Detailed theoretical analysis of the problem, with a running example, furnishes insights into the structure of the problem that has been instrumental in developing the algorithms. Further improvements to the algorithms are made using heuristic subroutines. These lead to an efficient and fast implementation that we demonstrate on a new dataset of road networks. We additionally demonstrate the application of our algorithms through two experiments on a campus road network.

Organization: The rest of the paper is organized as follows. The related work is discussed in Section 2. The single robot line coverage problem is formally described in Section 3. The section provides the ILP formulation and a continuous relaxation along with formal proofs. The approximation algorithms are developed in Section 4. The simulations and experiments are discussed in Section 5. Section 6 concludes the paper.

2 | RELATED WORK

The line coverage problem belongs to the broad class of arc routing problems (ARPs). A hierarchy of standard arc routing problems and the single robot line coverage problem is shown in Figure 2. The ARPs have traditionally been applied to transportation problems in which servicing is related to tasks such as delivery and pick up of goods [11]. Hence, the travel distances are used as costs and often have the same value whether the edge is serviced or deadheaded. Separate and asymmetric service costs are typically not considered.

Arc routing problems for a single vehicle: The Chinese postman problem (CPP) is to find an optimal tour such that every edge in a given undirected and connected graph is traversed at least once. Edmonds and Johnson [16] used matching and network flows to solve the CPP on undirected, directed, and Eulerian mixed graphs. The CPP on mixed or asymmetric graphs is an NP-hard problem. Frederickson [17] presented a $5/3$-approximation algorithm for the CPP on mixed graphs by using a combination of two approximation algorithms. The approximation factor was later improved to $3/2$ by Raghavachari and Veerasamy [35].

The asymmetric postman problem, also known as the windy postman problem (WPP), is the CPP with asymmetric edge costs. This problem is NP-hard, as shown by Guan [20]. Win [42] solved the WPP for Eulerian graphs in polynomial time by modeling it as a minimum cost flow problem. Win also designed a 2-approximation algorithm for WPP on general graphs

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

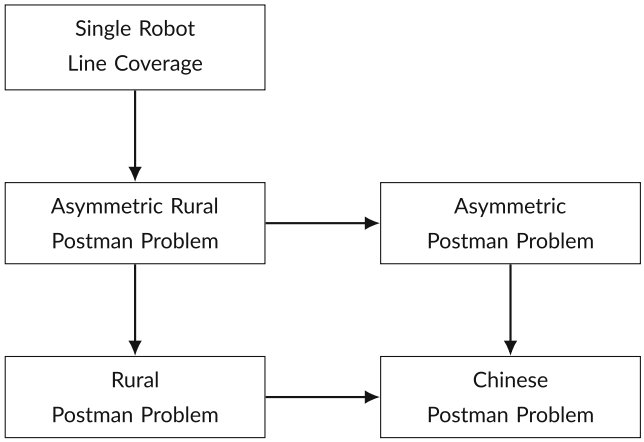


FIGURE 2 A hierarchy of arc routing problems with a single vehicle/robot. An arrow from problem A to problem B indicates that B is a special case of A. The single robot line coverage problem generalizes all the other depicted arc routing problems. Postman problems on asymmetric graphs are also termed *windy*, for example, the windy postman problem (WPP) and the windy rural postman problem.

using matching (to make the graph Eulerian) and minimum cost network flow. Raghavachari and Veerasamy [36] gave a $3/2$-approximation for the WPP. The CPP and the WPP do not allow non-required edges in the graph.

When the edges to be serviced are a subset of the edges in the graph, we have the rural postman problem (RPP). It was proved that the RPP is NP-hard by Lenstra and Kan [27]. For the RPP, Frederickson [17] gave a $3/2$-approximation algorithm similar to the algorithm by Christofides [9] for the metric traveling salesperson problem (TSP). The asymmetric RPP considers undirected graphs with asymmetric edge costs; that is, the cost of traversal of an edge can be different in the two directions. This problem is more commonly referred to as the windy rural postman problem in the arc routing literature. Lower bounds and heuristic algorithms were proposed by Benavent et al. [6,7] for the windy rural postman problem. However, these algorithms do not provide an approximation guarantee and have known pathological instances for which the ratio of the solution cost to the optimal cost is infinite. van Bevern et al. [40] showed that if the n-vertex asymmetric traveling salesperson problem (ATSP), subject to the triangle inequality constraint, is $\alpha(n)$-approximable in $t(n)$ time, then n-vertex RPP on an asymmetric and mixed graph is $(\alpha(C) + 3)$-approximable in $O(t(C) + n^3 \log n)$ time, where C is the number of weakly connected components in the subgraph induced by required arcs and edges. The single robot line coverage problem is closely related to the asymmetric RPP. However, in the asymmetric RPP, the costs of deadheading and servicing a required edge are the same. In contrast, the formulation in this paper allows distinct and asymmetric servicing and deadheading costs for required edges. The algorithms presented in this paper are applicable to the asymmetric RPP as any instance of the asymmetric RPP can be converted to that of the single robot line coverage problem by setting the cost of deadheading a required edge to the cost of the edge. The guarantee on the approximation factor for our algorithms is an improvement over the previously best-known algorithms given by van Bevern et al. [40] for the asymmetric RPP.

Exact and metaheuristic methods have been proposed for the ARPs, and they are covered in the survey paper by Corberán and Prins [12] and the monograph by Corberán and Laporte [11]. Exact methods include branch-and-cut with specific cutting plane procedures, branch-and-price, and column generation. One of the key techniques for these algorithms is to incorporate additional constraints that tighten the feasible space of the linear relaxation. Metaheuristic algorithms, such as scatter search, tabu search, and variable neighborhood descent, have also been used to solve the ARPs. However, these algorithms are not particularly suitable for robotics applications as they require significant computational resources. Moreover, they typically require a good initial solution as an additional input to upper bound the optimal cost of the instance. The algorithms presented in this paper can be used to provide such an initial solution with a guaranteed upper bound provided by the approximation factor.

Asymmetric traveling salesperson problem (ATSP): A dynamic programming algorithm that runs in $\mathcal{O}(n^2 2^n)$ computation time, where n is the number of vertices in the input graph, was given by Held and Karp [22] and Bellman [5]. The algorithm gives optimal solutions and can effectively solve small instances. Svensson et al. [38] were the first to present a constant-factor approximation algorithm for the ATSP with the triangle inequality. Traub and Vygen [39] improved the approximation ratio to $22+\epsilon, \epsilon > 0$ for the ATSP. These results are relevant for the theoretical guarantees on the approximation factor of our algorithms.

Line coverage in robotics: Line coverage has been used in robotics for the inspection of road networks and object boundaries. Dille and Singh [14] model the problem of road network coverage through tessellation of the road network by circles corresponding to the sensor footprint and finding a subset of the circular regions that covers the entire road network. Algorithms for node routing problems, such as the TSP and multiple TSP, are then used to find tours for the robots. Oh et al. [30] presented a mixed integer linear programming formulation and a heuristic algorithm for coverage of road networks using Dubins curves with Euclidean distances as costs. The nearest insertion heuristic method, originally designed for the TSP, is used to find

a sequence of edges to be visited while incorporating Dubins curves. The sequence is then split across a robot team using an auction algorithm. Easton and Burdick [15] proposed a constructive heuristic algorithm for the RPP with k vehicles for coverage of 2D object boundaries. The algorithm first groups the required edges into k clusters and computes a representative edge for each cluster. Additional edges are added to each cluster to ensure connectivity. Tours are computed for each cluster independently using the polynomial-time CPP algorithm proposed by Edmonds and Johnson [16]. Williams and Burdick [41] developed algorithms for boundary inspection while considering revisions to the path plan for the robots to account for the changes in the environment and in the robot team sizes. Xu and Stentz [45] use CPP and RPP formulations for line coverage and consider the case where the prior map information may be incomplete. They propose heuristic algorithms that can regenerate solutions rapidly when new map information is incorporated with the prior map. They extended this work to multiple robots [44] using k-means clustering to decompose networks into smaller components, similar to the algorithm presented by Easton and Burdick [15]. Campbell et al. [8] presented an application of ARPs to generate a route for a single UAV, where they allow the UAV to service a required edge in parts. The UAV may service a part of a required edge, move to some other edge, and come back later to service the remaining parts of the required edge. They convert the problem into standard ARPs by discretizing each required edge. The costs are considered to be Euclidean distances. Our algorithms are directly applicable to this discretized version of the problem.

These papers illustrate various applications of the line coverage problem. However, they do not consider asymmetric edge costs or distinct service and deadhead costs. Moreover, the heuristic algorithms do not provide theoretical guarantees on the quality of the solutions. The formulation and the algorithms presented in this paper address these shortcomings of the prior work.

Arc routing problems in area coverage: Arc routing problems have been used in robotics primarily as a subroutine in area coverage problems to generate efficient routes for a robot. Arkin et al. [4] use an algorithm similar to the one given by Edmonds and Johnson [16] for the CPP to find routes for a robot for the *milling problem*, a variant of the area coverage problem wherein the tool is constrained within the workspace. Mannadiar and Rekleitis [28] formulate the area to be covered in terms of edges in a Reeb graph. Optimal solutions to the CPP were used to compute an Euler tour for coverage of available free space while minimizing the path length. Karapetyan et al. [25] use the CPP formulation for k robots to find routes for multiple robots on a Reeb graph. They used the CPP to compute a large Euler tour and then break it into smaller tours using the algorithm given by Frederickson et al. [18]. Our algorithms for the single robot line coverage problem are directly applicable to the above techniques to generate routes for the area coverage problem.

3 | THE SINGLE ROBOT LINE COVERAGE PROBLEM

We now model the single robot line coverage problem as an optimization problem on a graph. We are given a connected undirected 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 set E can contain parallel edges between two vertices; that is, we allow G to be a multigraph. The service and deadhead costs are given as inputs along with the graph. The *single robot line coverage problem* is to find a coverage tour that minimizes the total cost of travel on the graph, such that each of the required edges in E_r is serviced exactly once.

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. If a robot *services* a required edge $e \in E_r$ in the direction a_e, then a service cost $c_s(a_e)$ is incurred; similarly for the direction $\bar{a}_e$. If a robot traverses an edge without servicing it, the robot is said to be *deadheading*; for example, this occurs when a robot is traveling from a vertex of an edge to that of another edge using a non-required edge. Both required and non-required edges may be deadheaded. Deadhead costs for an edge e are denoted by $c_d(a_e)$ and $c_d(\bar{a}_e)$. We use $c_s(A)$ and $c_d(A)$ to denote the corresponding sums of the service and deadhead costs for a set of arcs A. We denote by $\bar{A}$ the set of arcs oppositely directed to the arcs in A.

We consider the edge costs, for both servicing and deadheading, to be direction dependent; that is, the graph is asymmetric. For example, $c_s(a_e)$ may differ from $c_s(\bar{a}_e)$. The service and deadhead costs can be arbitrary positive numbers, with the constraint that the service cost of an edge is no less than the deadhead cost in the same direction. The costs, such as travel time, appear in the objective function of the problem. Since we allow edge costs to be asymmetric, the model allows both directed and mixed graphs. This can be achieved by setting the cost of the arcs that the robot is not allowed to travel to infinity or a very large constant. Additionally, we need to ensure that the graph is strongly connected so that there is a feasible solution. We also allow multiple copies of the edges and can model repeated servicing of segments.

3.1 | Preliminaries

Let $G = (V, E, E_r)$ be a connected undirected graph for the line coverage problem, such that $E_r \subseteq E$. The subgraph $G_r = (V_r, E_r)$ induced by the set of required edges E_r is called the *required graph* of G; $V_r \subseteq V$ is the set of vertices that have at least one

edge in E_r incident on them. The set of non-required edges is denoted by $E_n = E \setminus E_r$. We define the set of all arcs to be $\mathcal{A} = \bigcup\{a_e, \bar{a}_e\},\ \forall e \in E$. Similarly, $\mathcal{A}_r$ is defined for the set of required edges. If an arc a represents the travel direction from vertex u to vertex v, then the vertices u and v are called the tail $t(a)$ and head $h(a)$ of a, respectively. We denote by $H(\mathcal{A}, v)$ all the arcs $a \in \mathcal{A}$ that have v as the head. Similarly, $T(\mathcal{A}, v)$ is defined for the tail. The *degree* of a vertex $v \in V$ is the number of edges incident on v. A *walk* in a graph G is a non-empty alternating sequence $v_1e_1v_2e_2 \ \dots\ e_kv_{k+1}$ of vertices in V and edges in E such that v_i and v_{i+1} are the end vertices of an edge e_i for all $1 \leq i \leq k$. A *closed walk* is a walk with the same start and end vertex, that is, $v_1 = v_{k+1}$. An *Euler tour* is a closed walk such that every edge in the graph is traversed exactly once. A graph that has an Euler tour is called *Eulerian*. It is well established that an undirected graph is Eulerian if and only if every vertex has an even degree, see, for example, [34].

Let $D = (V, A)$ be a *directed graph* (digraph) with V as the set of vertices and A as the set of (directed) arcs. The digraph is strongly connected if there exists a directed path from any vertex in V to any other vertex in V. Analogous to the undirected graph, $T(A, v)$ and $H(A, v)$ are defined for the arc set A and a vertex $v \in V$. The *indegree* of a vertex $v \in V$, denoted by indeg(v), is the number of arcs entering the vertex v. Similarly, the *outdegree* of a vertex $v \in V$, denoted by outdeg(v), is the number of arcs going out of the vertex v. A digraph is Eulerian if and only if the graph is strongly connected and *balanced*, that is, indeg(v) = outdeg(v), $\forall v \in V$. *Imbalance* $\mathcal{I}(A, v)$ for the arc set A at a vertex v is given by outdeg(v) − indeg(v) = $|T(A, v)| - |H(A, v)|$. Analogous to the undirected graph, a *diwalk* is a sequence $v_1a_1v_2a_2 \ \dots\ a_kv_{k+1}$ of vertices and arcs in a digraph $D = (V, A)$ such that the tail of a_i is v_i and head of a_i is v_{i+1}. A *closed diwalk* is a walk with the same start and end vertices. An *Eulerian tour* on an Eulerian digraph is a closed diwalk such that each arc is traversed exactly once. An Euler tour can be constructed from an Eulerian graph (or digraph) in $\mathcal{O}(|A|)$ computational time, see, for example, [34].

Definition 1. Coverage Tour:

Given a connected graph $G = (V, E, E_r)$, a *coverage tour* is a closed walk in the graph G such that each required edge $e \in E_r$ is serviced exactly once.

Note that in a coverage tour, a required or a non-required edge may be used multiple times for deadheading. We define the following variables:

$$\begin{aligned} &s_{a_e}, s_{\bar{a}_e} \in \{0, 1\}, \ \text{ and } \ s_{a_e} + s_{\bar{a}_e} = 1 \quad \forall e \in E_r \\ &d_{a_e}, d_{\bar{a}_e} \in \mathbb{N} \cup \{0\} \quad \forall e \in E \end{aligned} \tag{1}$$

The variables s_{a_e} and $s_{\bar{a}_e}$ represent the two opposite directions of servicing the edge e; exactly one of the two can be equal to 1 for a valid coverage tour. The variables d_{a_e} and $d_{\bar{a}_e}$ represent the number of times an edge is deadheaded in the corresponding direction. The cost of a coverage tour τ is to be minimized and is given by:

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

For a valid coverage tour τ, we can create an Eulerian digraph $D_\tau = (V, A_\tau)$ from these variables by adding each arc as many times as the value of its corresponding variable. The digraph will have the same total cost as the coverage tour, that is, $c(\tau) = c(A_\tau)$. A closed diwalk can be obtained from D_τ in $\mathcal{O}(|A_\tau|)$ time. We will often use this equivalent Eulerian digraph representation of a coverage tour in the rest of the paper.

The following subsection presents an integer linear programming (ILP) formulation for the single robot line coverage problem. The ILP formulation allows us to formally define the problem in the form of an objective and a set of constraints, and provides an optimal solution. In subsequent sections, we provide a relaxation of the ILP formulation with continuous decision variables and relate the relaxation to a network flow model. The network flow model forms the basis of our approximation algorithms in Section 4.

3.2 | Integer linear programming formulation

The standard ILP formulations for arc routing problems involve an exponential number of constraints for ensuring connectivity [11] and are usually difficult to incorporate into standard ILP solvers. Therefore, we adopt the formulation presented by Gouveia et al. [19] as later specialized by Agarwal and Akella [1].

The primary decision variables of the formulation comprise the service variables s_{a_e} and $s_{\bar{a}_e}$ and the deadheading variables d_{a_e} and $d_{\bar{a}_e}$, as described in Section 3.1. The formulation needs an additional set of variables z_a, $\forall a \in \mathcal{A}$. These variables help in ensuring that the solution digraph computed by the ILP formulation is strongly connected, and therefore provides the corresponding coverage tour. The variables z_a can be interpreted as generalized flows across the arcs and are illustrated further in Theorem 1. Additionally, we select an arbitrary vertex $v_0 \in V_r$, a vertex corresponding to one of the ends of a required edge

and will be traversed in a coverage tour. This vertex acts as a source node for the generalized flow and is critical for proving the connectivity of the solution digraph.

SRLC-ILP (single robot line coverage ILP formulation)

Minimize:

$$c(\tau) = \sum_{e \in E_r} \left[c_s(a_e)\, s_{a_e} + c_s(\bar{a}_e)\, s_{\bar{a}_e} \right] + \sum_{e \in E} \left[c_d(a_e)\, d_{a_e} + c_d(\bar{a}_e)\, d_{\bar{a}_e} \right] \tag{3}$$

subject to:

$$\sum_{a_1 \in H(\mathcal{A}_r, v)} s_{a_1} + \sum_{b_1 \in H(\mathcal{A}, v)} d_{b_1} - \sum_{a_2 \in T(\mathcal{A}_r, v)} s_{a_2} - \sum_{b_2 \in T(\mathcal{A}, v)} d_{b_2} = 0, \quad \forall v \in V \tag{4}$$

$$s_{a_e} + s_{\bar{a}_e} = 1, \quad \forall e \in E_r \tag{5}$$

$$\sum_{a \in H(\mathcal{A}, v)} z_a - \sum_{a \in T(\mathcal{A}, v)} z_a = \sum_{a \in H(\mathcal{A}_r, v)} s_a, \quad \forall v \in V \setminus \{v_0\} \tag{6}$$

$$\sum_{a \in T(\mathcal{A}, v_0)} z_a = \sum_{a \in \mathcal{A}_r} s_a = |E_r| \tag{7}$$

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

$$z_a \leq |E_r| d_a, \quad \forall a \in \mathcal{A} \setminus \mathcal{A}_r \tag{9}$$

$$s_{a_e}, s_{\bar{a}_e} \in \{0, 1\}, \quad \forall e \in E_r \tag{10}$$

$$d_{a_e}, d_{\bar{a}_e} \in \mathbb{N} \cup \{0\}, \quad \forall e \in E \tag{11}$$

$$z_{a_e}, z_{\bar{a}_e} \in \mathbb{N} \cup \{0\}, \quad \forall e \in E \tag{12}$$

The objective function (3) minimizes the cost of a coverage tour. The *balance* (or symmetry) constraints (4) state that for each vertex, the number of arc traversals into a vertex should be equal to the number of arc traversals out of the vertex. The *servicing* constraints (5) ensure that each required edge is serviced exactly once and in only one direction. Constraints (6)–(9) are *connectivity* constraints and are a type of generalized flow constraints. The vertex v_0 is a source to a flow equal to the number of required edges $|E_r|$ as given by constraints (7). Constraints (6) state that a flow of one unit is absorbed each time a required edge is serviced. Building on the analysis by Gouveia et al. [19], we show that these constraints ensure that the solution digraph is connected in Lemma 1. The balance and the connectivity constraints together ensure that the resulting solution digraph has an Euler tour. The *integrality* constraints (10)–(12) ensure that the decision variables are integers. In the case of a mixed or a directed graph, where edges are not allowed to be traversed in one of the directions, we have an infinite[3] or a very large constant as the cost. If the solution digraph generated by the ILP formulation has such a high cost, the solution is treated as infeasible, and there is no corresponding coverage tour. An input graph with its optimal tour is shown in Figure 3.

We now prove the correctness of the ILP formulation by showing that the formulation gives an optimal coverage tour (Theorem 1). There are two components to the proof: (1) The solutions obtained from the SRLC-ILP formulation are Eulerian digraphs that correspond to feasible coverage tours (Lemma 1), and (2) Any feasible coverage tour has an equivalent feasible solution to the ILP (part of Theorem 1). Using the above two statements and the fact that the ILP formulation is an optimization problem with the cost of the coverage tour as the objective function, it follows that the optimal solution to the formulation has an equivalent optimal coverage tour.

Lemma 1. *Given a connected graph $G = (V, E, E_r)$, a solution to the SRLC-ILP formulation has an equivalent Eulerian digraph.*

Proof. A digraph is Eulerian if (a) each vertex in the digraph is balanced, and (b) the digraph is strongly connected. Given a solution to the SRLC-ILP formulation, we create a digraph $D_\epsilon = (V, A_\epsilon)$ with the same vertex set as the input graph and an arc set A_ϵ with $d_a + s_a$ copies of each arc a in $\mathcal{A}_r$ and d_a copies for each arc in $\mathcal{A} \setminus \mathcal{A}_r$. Note that the indegree equals the outdegree for each vertex in digraph D_ϵ because of balance constraints (4).

It remains to show that the digraph D_ϵ is strongly connected. In particular, we will show that any arc $a \in A_\epsilon$ with $s_a > 0$ and/or $d_a > 0$ is connected to an arbitrary vertex $v_0 \in V_r$, where V_r is the vertex set corresponding to the required graph. The vertex v_0 must be traversed by a feasible solution of the SRLC-ILP formulation as the vertex v_0 is connected to at least one of the required edges in E_r.

For our proof by contradiction, assume that D_ϵ is not connected, that is, it has more than one connected component. Consider one such connected component such that it is not connected to the vertex v_0. Since the solution

[3]The IEEE Standard for Floating-Point Arithmetic (IEEE 754) supports infinite values. MILP solvers, such as Gurobi, also provide the functionality to specify infinity.

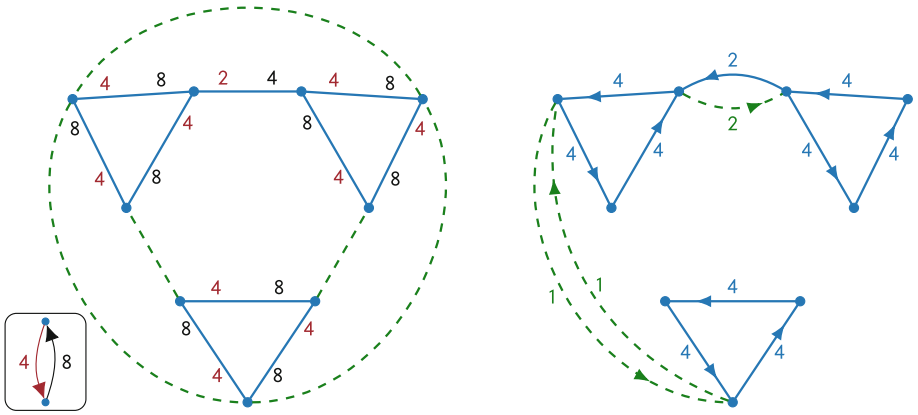


FIGURE 3 In the input graph (A), the solid blue lines and the dashed green lines represent required and non-required edges, respectively. The service costs in the two directions are shown in the input graph—the costs are closer to the head of the corresponding arc. Deadhead costs for the required edges are half the service costs in the respective directions. Non-required edges are set to have a unit cost in both directions. Deadheading costs are not shown. In the optimal coverage tour (B), the servicing arcs are shown as solid blue arcs, while the deadheadings are shown as green dashed arcs. The numbers in Figure (B) indicate the costs of the arcs in the final solution. The cost of the coverage tour is 42.

digraph is balanced, we can form an Eulerian diwalk that traverses *all* arcs in the selected connected component. We will assume that this Eulerian diwalk contains at least one arc that is being serviced, for otherwise, it is a diwalk of only deadheading arcs and can be eliminated without an increase in cost. Let $S \subset V$ be the set of vertices corresponding to this Eulerian diwalk such that $v_0 \notin S$. Define $\bar{S} = V \setminus S$. Note that $v_0 \in \bar{S}$.

Summing the constraints (6) over all the vertices in S gives the following equation:

$$\sum_{v \in S} \left(\sum_{a \in H(\mathcal{A}, v)} z_a - \sum_{a \in T(\mathcal{A}, v)} z_a \right) = \sum_{v \in S} \left(\sum_{a \in H(\mathcal{A}_r, v)} s_a \right) \tag{13}$$

For the purposes of this proof, define the following for any pair of sets $F, G \subseteq V$:

$$\begin{aligned} \delta(F, G) &= \{a \in A_\epsilon \mid t(a) \in F,\ h(a) \in G\} \\ R(F, G) &= \sum_{a \in \delta(F,G)} s_a, \quad N(F, G) = \sum_{a \in \delta(F,G)} d_a \\ Z(F, G) &= \sum_{a \in \delta(F,G)} z_a \end{aligned} \tag{14}$$

Then (13) can be written as:

$$Z(\bar{S}, S) - Z(S, \bar{S}) = R(S, S) + R(\bar{S}, S) \tag{15}$$

Note that $R(S, S) > 0$ and $R(\bar{S}, S) = 0$ from our assumption for contradiction. The term $Z(S, \bar{S})$ is non-negative. This implies that $Z(\bar{S}, S)$ is strictly positive.

Summing the constraints (8) and (9) over all arcs $a \in \delta(\bar{S}, S)$ gives:

$$Z(\bar{S}, S) \leq |E_r| \left(R(\bar{S}, S) + N(\bar{S}, S) \right) \tag{16}$$

Since $Z(\bar{S}, S) > 0$ and $R(\bar{S}, S) = 0$, it must be that $N(\bar{S}, S)$ is strictly positive. Thus, there is at least one arc with its tail in $\bar{S}$ and its head in S that is being deadheaded. There must also be a deadheading arc with its tail in S and its head in $\bar{S}$ because of the balance constraints (4). Hence, S and $\bar{S}$ are strongly connected, leading to a contradiction.

Thus, the digraph $D_\epsilon = (V, A_\epsilon)$ is balanced and strongly connected, that is, the digraph is Eulerian. A coverage tour can be obtained from the Eulerian digraph by computing an Eulerian diwalk in $\mathcal{O}(|A_\epsilon|)$ computation time. The cost of an Eulerian diwalk, and the corresponding coverage tour, on the digraph D_ϵ is $c(A_\epsilon) = c(\tau)$, where $c(\tau)$ is the value of the objective function for a solution to the SRLC-ILP formulation. A solution to the SRLC-ILP formulation has a corresponding feasible coverage tour. ■

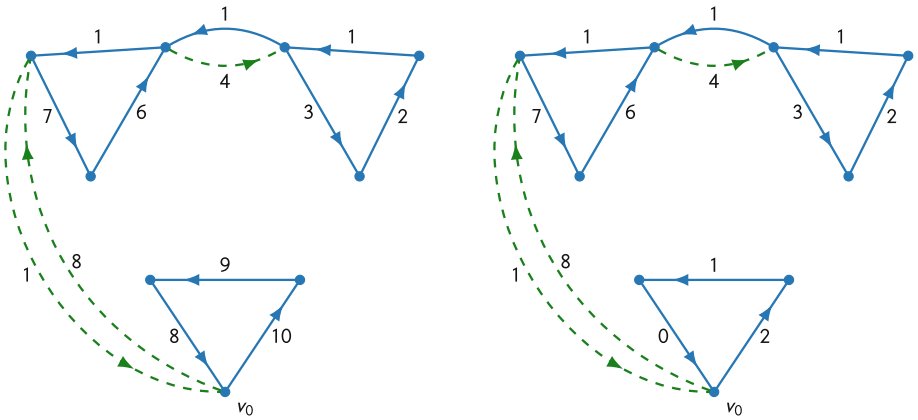


FIGURE 4 Two stages of assigning values of z_a to arcs from a given feasible coverage tour, as discussed in the proof of Theorem 1. The service and deadhead arcs are represented by solid and dashed lines, respectively. The arrows indicate the direction of travel. There are two loops connected to the depot vertex v_0. The numbers in the figures indicate values of z_a in the two stages, with the right figure showing the final values. The z values of the service arcs in the bottom triangle loop are reduced by 8. Note that $z_a = 0$ for all other arcs not shown in the digraph.

Theorem 1. *Given a connected graph $G = (V, E, E_r)$, the SRLC-ILP formulation gives an optimal coverage tour.*

Proof. We first prove that any feasible coverage tour τ has a corresponding feasible solution for the SRLC-ILP formulation with the same cost. In other words, the feasible solution space of the SRLC-ILP formulation includes all the feasible coverage tours. Represent the given coverage tour as a closed diwalk $v_1 a_1 \ \dots \ a_{i-1} v_i a_i \ \dots \ v_k a_k v_1$, along with information on whether an arc in the diwalk is serviced or deadheaded. For each variable $s_a, \forall a \in \mathcal{A}_r$ assign its value according to the direction in which the edge is serviced, and for each variable $d_a, \forall a \in \mathcal{A}$ assign its value equal to the number of times the arc is deadheaded. Also, let $D_\epsilon = (V, A_\epsilon)$ be the corresponding digraph. Note that $c(\tau) = c(A_\epsilon)$. Since the indegree of each vertex equals its outdegree in a connected closed diwalk, constraints (4) are satisfied. The diwalk must service each edge exactly once, satisfying (5).

Without loss of generality, assume that the first arc a_1 in the diwalk corresponds to a required edge; we can always perform a cyclic shift of the sequence of the arcs in the diwalk to obtain an equivalent diwalk that satisfies the assumption. Set $v_0 = v_1$, that is, the first vertex of the diwalk. Assume that the coverage tour visits v_0 only once; we will generalize this later. This implies that there are exactly two arcs in A_ϵ that are connected to v_0, one leaving and one entering. Set $z_a = 0$ for each arc in $\mathcal{A}$. Set $z_{a_1} = |E_r|$, satisfying constraint (7). Now traverse the edges in the sequence and direction given by the diwalk. Following the notation for a diwalk, arc a_i leaves vertex v_i. During the coverage tour traversal, if a vertex v_i is traversed for the first time, then set z_{a_i} to be $z_{a_{i-1}}$ minus the number of arcs that are marked for servicing and enter the vertex v_i. If v_i has already been traversed by an arc whose tail is v_i, then set $z_{a_i} = z_{a_i} + z_{a_{i-1}}$. The value of z_a remains zero for the arcs that were not traversed by the diwalk. Thus, constraints (8) and (9) are satisfied for all the arcs. The constraints (6) will be satisfied at each vertex, other than v_0, by construction.

Now we address the case when the coverage tour traverses the vertex v_0 multiple times. This will result in l loops at v_0, which we index by $j = 1, \dots, l$. We first perform the same procedure to assign the values of z_a as in the preceding paragraph. Let the first arc (leaving v_0) and last arc (entering v_0) for a loop j be a_{j_1} and a_{j_k}, respectively. Now for each loop j, reduce the value of z for all arcs in the loop j by the value of z_a for the last arc in the loop $a = a_{j_k}$. Finally, increase the value of z for all arcs in any one of the loops by $|E_r| - \sum_{j \in \{1, \dots, l\}} z_{a_{j_1}}$, to satisfy (7). An example is shown in Figure 4.

Hence, we can compute a feasible solution to the SRLC-ILP formulation from a feasible connected closed diwalk with the same cost, that is, the feasible solution space has all the feasible closed diwalks (coverage tours). From Lemma 1, an optimal solution to the SRLC-ILP formulation gives a feasible connected closed diwalk of the same cost. As the objective function of the SRLC-ILP formulation corresponds to the cost of a coverage tour, an optimal feasible solution to the formulation will also be an optimal coverage tour for the single robot line coverage problem. Hence, the SRLCILP formulation gives an optimal coverage tour for a connected graph. ∎

3.3 | Continuous relaxation of SRLC-ILP

We consider a continuous relaxation of the SRLC-ILP formulation that is closely related to the minimum cost flow problem. To obtain this relaxation, we remove the connectivity constraints and relax the integer variables to make them continuous. The relaxation, as we shall see later, gives insights into the structure of the problem and enables the development of the approximation

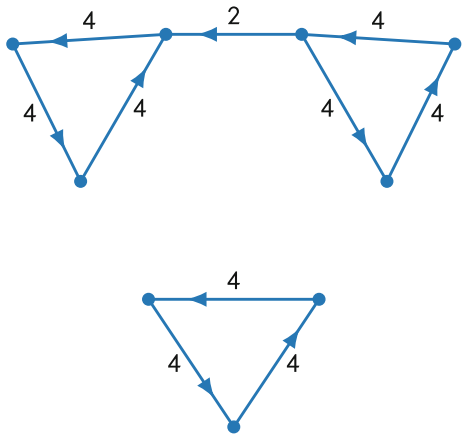


FIGURE 5 The min-cost digraph $D_m = (V, A_m)$ for the input graph given in Figure 3. Note that the graph is neither balanced nor connected.

algorithms. The key idea is to first build a min-cost digraph, which is not necessarily balanced, and then use an LP formulation, modeled as a minimum cost flow problem, to reverse some of the service arcs in the digraph and add deadheading arcs such that the resulting digraph is balanced. We will use the flow problem to develop approximation algorithms for the different cases of the single robot line coverage problem based on the structure of the required graph.

As before, let the input graph be $G = (V, E, E_r)$ with a required graph $G_r = (V_r, E_r)$. Modify the SRLC-ILP formulation as follows:

1. Remove the connectivity constraints (6)–(9) from the SRLC-ILP formulation. When the required graph G_r is connected, the ILP formulation is still valid.
2. Generate a digraph $D_m = (V, A_m)$ using the algorithm MinCostDigraph (Algorithm 1), which selects the arc with the minimum service cost for each required edge. The min-cost digraph D_m for an input graph is shown in Figure 5.
3. Introduce a *reverse* variable r_a for each arc $a \in A_m$, to represent the reversal of service direction of the arc a. The reverse variables r_a take values from $\{0, 2\}$; $r_a = 0$ when the service direction is not changed, and $r_a = 2$ when the direction is reversed.
4. Relax the integrality constraints (11) so that the deadheading variables $d_{a_e}, d_{\bar{a}_e}$, and the new reverse variables r_a are now continuous.

If an arc's service direction is reversed from a to $\bar{a}$, $r_a = 2$, the imbalance changes by 2, and the total cost changes by $c_s(\bar{a}) - c_s(a)$. We assign *reversal cost* $c_r(a)$ for each arc $a \in A_m$ and set $c_r(a)$ to $\frac{c_s(\bar{a}) - c_s(a)}{2}$.

We now state the continuous relaxation of the SRLC-ILP formulation.

SRLC-LP (single robot line coverage linear programming relaxation)

Minimize:

$$c_s(A_m) + \sum_{a \in A_m} c_r(a)\, r_a + \sum_{e \in E} \left[c_d(a_e)\, d_{a_e} + c_d(\bar{a}_e)\, d_{\bar{a}_e} \right] \tag{17}$$

subject to:

$$\sum_{a_1 \in H(A_m, v)} - r_{a_1} + \sum_{b_1 \in H(\mathcal{A}, v)} d_{b_1} + \sum_{a_2 \in T(A_m, v)} r_{a_2} - \sum_{b_2 \in T(\mathcal{A}, v)} d_{b_2} = \mathcal{I}(A_m, v), \quad \forall v \in V \tag{18}$$

$$0 \le r_a \le 2, \quad \forall a \in A_m \tag{19}$$

$$d_{a_e}, d_{\bar{a}_e} \ge 0, \quad \forall e \in E \tag{20}$$

Expression (17) is the modified objective function. The first term in the objective function $c_s(A_m)$ is the sum of the service costs of all the arcs in the digraph D_m and is independent of the variables. The imbalance in the digraph D_m at a vertex v is represented by $\mathcal{I}(A_m, v)$. The conditions (18) ensure that the digraph corresponding to a feasible solution will be balanced, that is, the indegree will equal the outdegree at every vertex. The constraints (19) state that the variable r_a, corresponding to the reversal of service direction, is between 0 and 2; if $r_a = 0$, the direction of travel is the same as that of the arc in A_m, and if $r_a = 2$, the direction of the arc is reversed, thereby reversing the service direction.

ALGORITHM 1. MINCOSTDIGRAPH

Input : Graph $G = (V, E, E_r)$
Output: Minimum cost digraph $D_m = (V, A_m)$

```
A_m ← ∅;
for e ∈ E_r do
    if c_s(a_e) ≤ c_s(ā_e) then
        a_e.SERVICE ← TRUE; A_m.INSERT(a_e);
    else
        ā_e.SERVICE ← TRUE; A_m.INSERT(ā_e);
    end
end
```

TABLE 1 Flow model (FM): Arc costs and capacities.

Arc	Description	Unit flow cost $c_f(\cdot)$	Capacity
a	Forward deadheading	$c_d(a)$	∞
$\bar{a}$	Backward deadheading	$c_d(\bar{a})$	∞
a'	Service reversal	$c_r(a') = (c_s(\bar{a}) - c_s(a))/2$	2
b	Non-required forward deadheading	$c_d(b)$	∞
$\bar{b}$	Non-required reverse deadheading	$c_d(\bar{b})$	∞

Note: Three arcs $(a, \bar{a}, a')$ are added for each required edge, and two arcs $(b, \bar{b})$ for each non-required edge.

3.4 | A network flow graph model

Arc routing problems are often solved by modeling them as network flow graphs and finding a minimum cost flow. Using this approach, algorithms for the CPP and the WPP were presented by Edmonds and Johnson [16] and Win [42], respectively. Inspired by such techniques, we present a network flow graph model for solving the linear programming formulation SRLC-LP and establish its equivalence. We will then use the model to develop approximation algorithms in Section 4.

Let $G = (V, E, E_r)$ be the input graph. First, generate a min-cost digraph $D_m = (V, A_m)$ using the algorithm MINCOSTDIGRAPH(G). Now construct a network flow graph $D_f = (V, A_f)$, in $\mathcal{O}(|V| + |E|)$ time (Algorithm 2), as follows:

1. For each service arc $a \in A_m$, add three arcs a, $\bar{a}$, and a' to A_f with the costs per unit flow $c_f(\cdot)$ and capacities as given in Table 1, which defines the Flow Model. The direction of arc a is the same as that in A_m, whereas the direction of arcs $\bar{a}$ and a' are opposite to that of the corresponding arc in A_m. In the flow digraph D_f, the arcs a and $\bar{a}$ will represent deadheadings, and the arc a' will represent service reversal.
2. For each non-required edge $e_n \in E \setminus E_r$, add two arcs b and $\bar{b}$ to A_f, with the costs per unit flow and capacities in Table 1. These two arcs, b and $\bar{b}$, represent deadheadings of a non-required edge in the two directions.
3. For each vertex $v \in V$, assign the following node flow demand based on the degree of v in D_m:

$$d(v) = \text{outdeg}(v) - \text{indeg}(v) = \mathcal{I}(A_m, v) \tag{21}$$

Let f_a be the flow along the arc $a \in A_f$, and let the *flow vector* be $\mathbf{f} = [f_a \mid a \in A_f]$. The cost $c(\mathbf{f})$ of a flow $\mathbf{f}$ is:

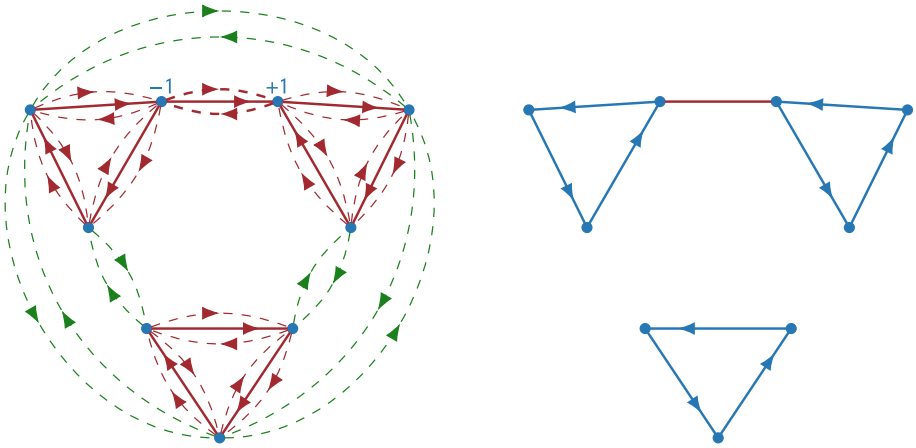
$$c(\mathbf{f}) = \sum_{a \in A_f} c_f(a)\, f_a \tag{22}$$

A flow digraph D_f is shown in Figure 6A for the input graph of Figure 3, with the min-cost digraph shown in Figure 5.

We now formulate a minimum cost flow problem for the network flow graph $D_f = (V, A_f)$ and show that it models the continuous relaxation SRLC-LP.

Definition 2. Minimum cost flow problem:

Let $D_f = (V, A_f)$ be a given flow digraph, along with costs, capacities, and node flow demands. Then the minimum cost flow problem is to find a feasible flow $\mathbf{f}$ such that:



(A) Flow digraph $D_f = (V, A_f)$ (B) Optimal solution to SRLC-LP with ambiguous (undirected) edges

FIGURE 6 Flow digraph and an optimal solution to the SRLC-LP problem for the input graph G in Figure 3 with the min-cost digraph shown in Figure 5. (A) The red (darker) solid arcs correspond to service reversal arcs, and the red dashed arcs correspond to the deadheading of required edges. The dashed green (lighter) arcs are flow arcs corresponding to deadheading across non-required edges. The nonzero imbalances for the vertices are shown. (B) The set of arcs corresponding to ambiguous edges A_u—those that remain undirected after solving the flow problem—is shown in red (darker).

ALGORITHM 2. CONSTRUCTFLOWDIGRAPH

Input : Graph $G = (V, E, E_r)$, Digraph $D_m = (V, A_m)$
Output: Flow Digraph $D_f = (V, A_f)$

$A_f \leftarrow \emptyset$;
for $a \in A_m$ **do**
  Insert arcs a, $\bar{a}$ and a' into A_f, with costs and capacities from Table 1;
end
for $e \in E \setminus E_r$ **do**
  Let b_e and $\bar{b}_e$ be the arcs corresponding to e;
  Insert arcs b_e and $\bar{b}_e$ into A_f, with costs and capacities from Table 1;
end
for $v \in V$ **do**
  $d(v) \leftarrow \mathcal{I}(A_m, v)$;
end

1 the cost of flow $c(\mathbf{f})$ is minimized, and
2 demand $d(v)$ is satisfied for all $v \in V$.

Theorem 2. *Let $G = (V, E, E_r)$ be an input graph for the single robot line coverage problem, with minimum cost digraph $D_m = (V, A_m)$ and flow digraph $D_f = (V, A_f)$. The minimum cost flow problem for network flow digraph D_f models the continuous relaxation SRLC-LP of the SRLC-ILP formulation for the single robot line coverage problem.*

Proof. Observe that any feasible solution to the minimum cost flow problem is a feasible solution to the linear programming formulation SRLC-LP, and vice versa, using the following relation between the variables:

$$\begin{aligned} f_a = d_a,\ f_{a'} = r_a,\ f_{\bar{a}} = d_{\bar{a}}, \qquad & \forall a \in A_m \\ f_{a_e} = d_{a_e},\ f_{\bar{a}_e} = d_{\bar{a}_e}, \qquad & \forall e \in E \setminus E_r \end{aligned} \tag{23}$$

As the capacity of the reversal arcs $r_{a'}$ is set to 2, the flow $f_{a'}$ across any such arc will be no greater than 2, and the constraints (19) are satisfied. The flow problem resolves the demand $d(v) = \mathcal{I}(A_m, v)$ for each vertex $v \in V$, and thus, satisfies the balance constraints (18). The objective of the minimum cost flow problem $c(f)$ summed with $c_s(A_m)$ is then exactly the objective function (17) of SRLC-LP. Thus, the theorem follows. ∎

Remark 1. The minimum cost flow for a graph $G = (V, E)$ can be computed in time $\mathcal{O}((m \log n)(m + n \log n))$, as shown by Orlin [32], where $m = |E|$ and $n = |V|$. Optimal solutions to the minimum cost flow algorithms are integers as the imbalance at each vertex is also an integer (see, e.g., [34]), thus $f_{a'} \in \{0, 1, 2\}$. When we set $r_a = f_{a'}$ we may have $r_a = 1$. This, in turn, violates the integrality constraints (10), and the corresponding service variables are half-integral: $s_a = s_{\bar{a}} = 0.5$. For any coverage tour, there exists an equivalent network flow digraph with a corresponding flow. However, not all solutions to the network flow digraph have an equivalent coverage tour.

4 | APPROXIMATION ALGORITHMS

We now present approximation algorithms for the single robot line coverage problem by partitioning it into three cases, as illustrated in Figure 7. The cases are based on the structure of the required graph, that is, the subgraph induced by only the required edges:

1. Eulerian required graph: The algorithm LP-SOLVE, derived from the network flow model, gives an optimal solution.
2. Connected required graph: The 2-approximation algorithm SRLC-2APPROX gives a solution with a cost at most twice the optimal cost.
3. General required graph: The $(\alpha(C) + 2)$-approximation algorithm, where C is the number of connected components in the required graph, and $\alpha(C)$ is the approximation factor for the ATSP on a graph with C vertices, gives a solution with cost at most $\alpha(C) + 2$ times the optimal cost. When the number of connected components in the required graph is small, that is, $C \in \mathcal{O}(\log(n))$, a 3-approximation solution is obtained.

We recast the theoretical results of the previous section in algorithmic form as Algorithm 3, which computes a digraph by solving the linear relaxation to the single robot line coverage problem and creates a balanced digraph $D_b = (V, A_b)$. Given a graph $G = (V, E, E_r)$, we first compute the min-cost digraph $D_m = (V, A_m)$ (line 1) and construct the flow digraph $D_f = (V, A_f)$ from D_m (line 2). We then compute the minimum cost flow $\mathbf{f}$ for the flow digraph (line 3). If the optimal flow across an arc a is 0 or 2, we set the service direction of the corresponding edge to the direction of a or $\bar{a}$, respectively, and add the corresponding arc to the arc set A_b (lines 6–11). For some of the required edges, the optimal flow through the reversal arcs can be 1, that is, $r_a = 1$. These arcs, denoted by A_u, correspond to the edges whose service direction remains *ambiguous* in the solution to the flow problem (lines 12–13). Finally, we add deadheading arcs to the arc set A_b according to the corresponding optimal flow through the deadheading arcs of the flow digraph (lines 14–20). The digraph $D_b = (V, A_b)$ and the set of ambiguous arcs A_u are the output of the algorithm. Note that the digraph D_b is balanced but may have multiple components, each of which is Eulerian. An output of LP-SOLVE for the input graph given in Figure 3 is shown in Figure 6B. The arcs corresponding to the digraph D_b are shown in blue, whereas the ambiguous edge, corresponding to the edge set A_u, is shown in dark red.

4.1 | Eulerian required graph

We first consider the case where the required graph $G_r = (V_r, E_r)$, for the input graph $G = (V, E, E_r)$, is Eulerian, that is, the subgraph is connected, and the degree of each vertex in V_r is even. It should be noted that this special case is not the same as the Eulerian graphs for the asymmetric/windy postman problem (WPP) [36,42]. This is because non-required edges are permitted, and the deadheading costs for the required edges can differ from their service costs. Thus, an Eulerian tour on the required graph does not ensure a coverage tour with minimum cost. The following lemma shows that the LP-SOLVE algorithm gives an optimal solution for this case in running time that is polynomial in the number of edges and vertices.

Theorem 3. *Let the input graph be $G = (V, E, E_r)$ such that the required graph $G_r = (V_r, E_r)$ is Eulerian. Then algorithm* LP-SOLVE *gives an optimal feasible solution for the single robot line coverage problem in polynomial time. In particular, $r_a = f_{a'} \in \{0, 2\} \quad \forall a \in A_m$ and $d_{a_e} = f_{a_e}$, $d_{\bar{a}_e} = f_{\bar{a}_e} \in \mathbb{N} \cup \{0\} \quad \forall e \in E$.*

Proof. Let the min-cost digraph be $D_m = (V, A_m)$ and the flow digraph be $D_f = (V, A_f)$, for the input graph $G = (V, E, E_r)$. Since the required graph G_r is Eulerian, the number of edges incident at a vertex in G_r is even. The number of outgoing arcs and incoming arcs, at a vertex in D_m, will both be even or will both be odd. Therefore, the node flow demand for a vertex $v \in V$ computed for the digraph D_m will be even. The capacities defined in Table 1 are either 2 or ∞. Furthermore, the minimum cost flow algorithm gives integral solutions. Hence, the optimal flow is also even for each arc $a_f \in A_f$ and, in particular, $r_a = f_{a'} \in \{0, 2\}$. This result can also be derived by establishing the total unimodularity of the constraint matrix obtained from the balance constraints (18) by replacing each r_a by

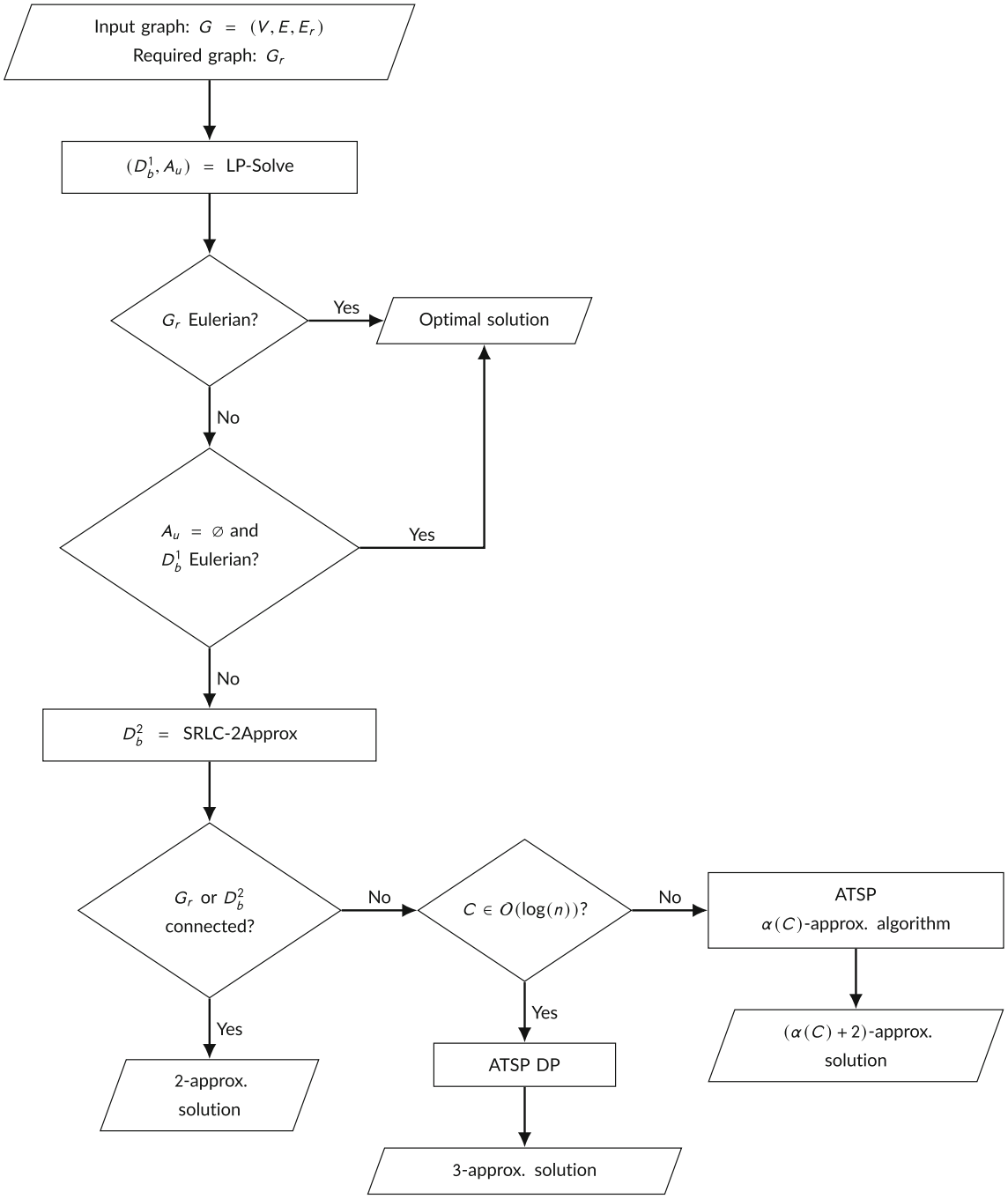


FIGURE 7 Flowchart illustrating the different cases of the single robot line coverage problem based on the structure of the required graph G_r. The approximation factors of the algorithms depend on the structure of G_r. The approximation factor for the most general case is $(\alpha(C) + 2)$, where C is the number of connected components in the required graph, and $\alpha(C)$ is the approximation factor for an ATSP algorithm on a graph with C vertices.

$2\tilde{r}_a$, d_a by $2\tilde{d}_a$, and $d_{\bar{a}}$ by $2\tilde{d}_{\bar{a}}$. Note that the imbalance $\mathcal{I}(A_m, v)$ is even for each vertex $v \in V$. Thus, we can divide the entire equation by 2, and the constraints will still have integral coefficients and constants.

$$\sum_{a_1 \in H(A_m, v)} -\tilde{r}_{a_1} + \sum_{b_1 \in H(\mathcal{A}, v)} \tilde{d}_{b_1} + \sum_{a_2 \in T(A_m, v)} \tilde{r}_{a_2} - \sum_{b_2 \in T(\mathcal{A}, v)} \tilde{d}_{b_2} = \frac{\mathcal{I}(A_m, v)}{2} \quad \forall v \in V$$

The constraint matrix for the above equation corresponds to an incidence matrix with integers, and thus is totally unimodular. The substituted variables will be integral, and the original variables will all be even. As the required graph is connected, the digraph D_b obtained from the algorithm will also be connected, and thus will be Eulerian.

Let $m = |E|$ and $n = |V|$. The min-cost digraph and the flow digraph can be constructed in $\mathcal{O}(n)$ and $\mathcal{O}(m + n)$ time, respectively. Thus, the running time of the algorithm is dependent on the algorithm for solving the minimum cost flow problem, that is, $\mathcal{O}((m \log n)(m + n \log n))$. ■

ALGORITHM 3. LP-SOLVE

Input : Graph $G = (V, E, E_r)$
Output: Digraph $D_b = (V, A_b)$ and undirected arc set A_u

$D_m = (V, A_m) \leftarrow$ MINCOSTDIGRAPH(G);
$D_f = (V, A_f) \leftarrow$ CONSTRUCTFLOWDIGRAPH(G, D_m);
Compute the minimum cost flow $\mathbf{f}$ for D_f;
/* Generate digraph D_b */
$A_b \leftarrow \emptyset$;
for $a \in A_m$ **do**
  if $f_{a'} = 0$ **then**
    a.SERVICE $\leftarrow$ TRUE;
    A_b.INSERT(a);
  else if $f_{a'} = 2$ **then**
    $\bar{a}$.SERVICE $\leftarrow$ TRUE;
    A_b.INSERT($\bar{a}$);
  else if $f_{a'} = 1$ **then**
    A_u.INSERT(a);
  A_b.INSERT(f_a copies of a);
  A_b.INSERT($f_{\bar{a}}$ copies of $\bar{a}$);
end
for $e \in E \setminus E_r$ **do**
  A_b.INSERT(f_{a_e} copies of a_e);
  A_b.INSERT($f_{\bar{a}_e}$ copies of $\bar{a}_e$);
end

4.2 | Connected required graph

We now consider the case where the required graph $G_r = (V_r, E_r)$, for the input graph $G = (V, E, E_r)$, is connected but not necessarily Eulerian, that is, the degree of each vertex in G_r might not be even. The created flow digraph $D_f = (V, A_f)$ may have vertices with odd flow demands (21). As a result, the optimal flow values need not be even. While it is not a problem if d_a or $d_{\bar{a}}$ is odd for some arc $a \in \mathcal{A}$, we need to assign service directions to the edges for which the reverse variables r_a is 1 for arcs $a \in A_m$ and potentially add deadheading arcs to make the corresponding digraph Eulerian.

Let A_u be the set of arcs for which the flow for the arc corresponding to the reversal of service direction is 1, that is, $A_u = \{a \mid r_a = f_{a'} = 1, a \in A_m\}$, as given by algorithm LP-SOLVE, and let E_u be the corresponding edge set. We first check for cycles in the graph (V, E_u). Such cycles will have a sequence of edges such that the total cost of the edges, if oriented in the clockwise direction, is the same as the total cost of the edges if oriented in the anti-clockwise direction since the flow obtained is optimal. We can orient such cycles in either clockwise or anti-clockwise order without changing the cost of the solution. Now, let us say we service the edge corresponding to some $a \in A_u$ in the same direction as a. This creates an imbalance of $+1$ at the tail $t(a)$ and -1 at the head $h(a)$. We can resolve this by adding the shortest deadheading path from $h(a)$ to $t(a)$, the cost of which is denoted by $c_d(h(a), t(a))$. The total cost due to traversals of this edge will be the sum of the service cost of a and this deadheading cost. Similarly, we can consider servicing in the direction of $\bar{a}$ and then consider deadheading from $h(\bar{a})$ to $t(\bar{a})$. Of the two combinations, the one that has a lower total cost is selected. This is done for each ambiguous edge corresponding to $a \in A_u$. This idea is described concretely in algorithm SRLC-2APPROX. Raghavachari and Veerasamy [36] used a linear relaxation to get a lower bound on the optimal cost for the WPP. Using a similar lower bounding approach, we show that the algorithm SRLC-2APPROX (Algorithm 4) computes a coverage tour with a cost at most twice that of the optimal solution.

Let the optimal flow be $\mathbf{f}$, and the cost of the optimal tour be c^*. Then the optimal value of the linear relaxation z^* of the formulation SRLC-LP is:

$$z^* = c_s(A_m) + c(\mathbf{f}) = c_s(A_m \setminus A_u) + c_s(A_u) + c(\mathbf{f}) \leq c^* \tag{24}$$

Let A_d denote the set of arcs with service direction and deadheading decided unambiguously by the optimal flow. Then,

$$c(A_d) = c_s(A_m \setminus A_u) + c(\mathbf{f}) - c_r(A_u) \tag{25}$$

ALGORITHM 4. SRLC-2APPROX

Input : Graph $G = (V, E, E_r)$
Output: Digraph $D_b = (V, A_b)$
$(D_b^{\text{LP}} = (V, A_b^{\text{LP}}), A_u) \leftarrow$ LP-SOLVE(G);
/* Let E_u be the edge set corresponding to A_u */
$A_b \leftarrow A_b^{\text{LP}}$;
Find cycles in the graph (V, E_u);
Orient cycles in anti-clockwise orientation and add arcs to A_b;
for $a \in A_u$ **do**
  $p \leftarrow$ shortest deadheading path from $h(a)$ to $t(a)$;
  $\bar{p} \leftarrow$ shortest deadheading path from $h(\bar{a})$ to $t(\bar{a})$;
  if $c_s(a) + c_d(p) \leq c_s(\bar{a}) + c_d(\bar{p})$ **then**
    a.SERVICE $\leftarrow$ TRUE;
    A_b.INSERT(a);
    A_b.INSERT(p);
  else
    $\bar{a}$.SERVICE $\leftarrow$ TRUE;
    A_b.INSERT($\bar{a}$);
    A_b.INSERT($\bar{p}$);
  end
end

Thus,

$$z^* = c(A_d) + c_s(A_u) + c_r(A_u) \tag{26}$$

Substituting the value of $c_r(A_u) = \frac{c_s(\bar{A}_u) - c_s(A_u)}{2}$ in (26) and using (24), we have,

$$\begin{aligned} 2c(A_d) + c_s(A_u) + c_s(\bar{A}_u) &\leq 2c^* \\ \text{or, } c(A_d) + c_s(A_u) + c_s(\bar{A}_u) &\leq 2c^* - c(A_d) \end{aligned} \tag{27}$$

Theorem 4. *Let the input graph be $G = (V, E, E_r)$ such that the required graph $G_r = (V_r, E_r)$ is connected. Then algorithm* SRLC-2APPROX *generates a coverage tour with cost at most twice the cost of the optimal coverage tour in polynomial time.*

Proof. Let A_s be the set of arcs corresponding to A_u with final service directions as oriented by the algorithm SRLC-2APPROX. The total cost of the solution digraph $D_b = (V, A_b)$ generated by the algorithm is:

$$c(A_b) = c(A_d) + c_s(A_s) + \sum_{a \in A_s} c_d(h(a), t(a)) \leq c(A_d) + c_s(A_u) + c_d(\bar{A}_u)$$

Note that the inequality is true because we selected the service and deadheading directions to minimize the sum of the costs for individual arcs in A_u.

Furthermore, $c_d(a) \leq c_s(a)$ for $a \in \bar{A}_u$ because the deadheading cost is assumed to be no greater than the corresponding service cost. Hence,

$$c(A_b) \leq c(A_d) + c_s(A_u) + c_s(\bar{A}_u) \tag{28}$$

Combining (28) with (27):

$$c(A_b) \leq 2c^* - c(A_d) \leq 2c^*$$

As the required graph G_r is connected, the solution digraph D_b is also connected. The digraph D_b is also balanced, as discussed previously. Hence, a coverage tour can be generated by computing an Eulerian diwalk on D_b with the same cost as that of A_b. Thus, we obtain a coverage tour of cost at most twice the cost of the optimal tour.

The complexity of the algorithm is determined by the algorithm LP-SOLVE, which can be solved in $\mathcal{O}((m \log n)(m + n \log n))$ time, where $m = |E|$ and $n = |V|$. Depending on the structure of 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 $\mathcal{O}(n^3)$ computation time, see, for example, [13]. ■

4.3 | General required graph

We now consider input graphs for which the required graph G_r, induced by the required edges, may have multiple connected components. For such graphs, algorithm SRLC-2APPROX may output a disconnected digraph even though the individual connected components are Eulerian. For the graph given in Figure 3, with the flow digraph given in Figure 6, the output of algorithm SRLC-2APPROX is shown in Figure 8. Note that the digraph has multiple connected components even though the individual components are Eulerian.

Our approach is to generate a tour through the connected components by solving the ATSP problem on an auxiliary graph whose vertices correspond to the components. We combine the tour with the arcs generated in each component. We develop an $(\alpha(C) + \beta)$-approximation algorithm where C is the number of connected components in G_r and β is the approximation factor for the single robot line coverage problem on graphs with a connected required graph. The α approximation factor depends on the approximation algorithm for the asymmetric traveling salesperson problem (ATSP), and a β of 2 was discussed in the previous subsection using the SRLC-2APPROX algorithm. Constant factor approximation algorithms for ATSP were recently given by Svensson et al. [38] and by Traub and Vygen [39].

The output digraph D_b of the SRLC-2APPROX algorithm is processed to find strongly connected components. Note that the number of strongly connected components in D_b will be no greater than the number of connected components C of the required graph G_r of G. We then create an auxiliary graph $G_0 = (V_0, E_0)$ with $V_0 \subseteq V_r$ consisting of one arbitrary vertex from each connected component in D_b such that a vertex $v \in V_0$ corresponds to a required edge. The proof for the approximation factor, as we will see in Theorem 5 below, is agnostic to the choice of the required vertex in each connected component to form the vertex set V_0. We add an edge for each pair of vertices in V_0 to the edge set E_0. For each edge $e \in E_0$, we assign two weights corresponding to the shortest deadhead cost path between the vertices of the edge in the two directions. This makes the graph G_0 complete with asymmetric edge costs. An ATSP algorithm is then used to find a tour connecting all the vertices in G_0. The arcs in the ATSP tour are then added to the disconnected diwalk generated from the SRLC-2APPROX algorithm to obtain a connected coverage tour. In the following theorem, we prove the approximation factor for our algorithm for general graphs. The key ideas for the proof of the theorem are motivated by [40].

Theorem 5. *The single robot line coverage problem can be solved in polynomial time with an approximation factor of $\alpha(C) + \beta$, where $\alpha(C)$ is the approximation factor for an algorithm for the asymmetric traveling salesperson problem with C vertices, and β is the approximation factor for line coverage on graphs with a connected required graph.*

Proof. Let τ^* be the optimal coverage tour and digraph D_b be the output of the SRLC-2APPROX algorithm. Note that D_b may contain multiple strongly connected components. However, the number of strongly connected components in the digraph D_b will be no more than the number of connected components C in the required graph G_r. As the linear relaxation does not consider the connectivity constraints, the solution D_b is an approximation result to a relaxation of the original problem. Hence, $c(D_b) \leq \beta\ c(\tau^*)$, where $\beta = 2$ for the SRLC-2APPROX algorithm.

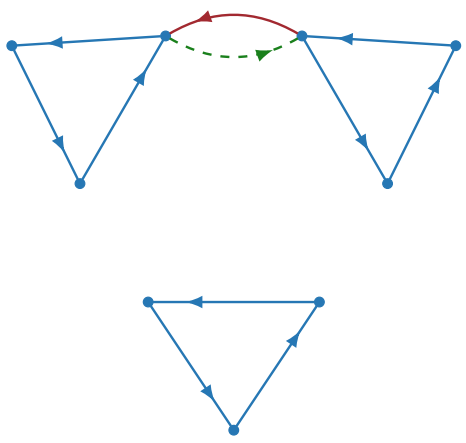


FIGURE 8 Digraph computed by the algorithm SRLC-2APPROX for the graph in Figure 3 with the flow digraph shown in Figure 6. Note that the digraph has multiple connected components, each of which is Eulerian.

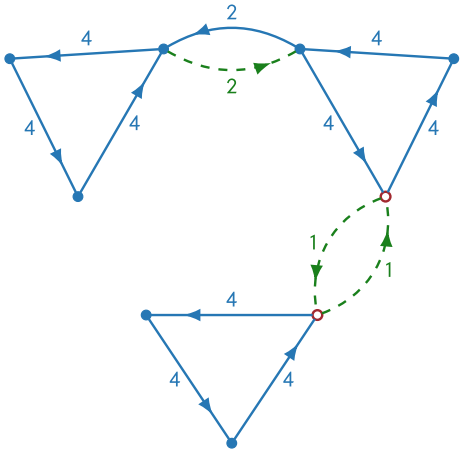


FIGURE 9 The final coverage tour, in the form of an Eulerian digraph, obtained for the input graph in Figure 3. The algorithm generates an ATSP tour on the solution from SRLC-2APPROX, shown in Figure 8. For each connected component, an arbitrary vertex is selected, shown as red unfilled circles. As there are two connected components, the algorithm creates an auxiliary vertex by duplicating one of the selected vertices. A different choice for these vertices can result in a different coverage tour, potentially of a different cost. The total cost of the coverage tour is 42, which is the same as the optimal cost.

Let $V_0 \subseteq V_r$ be a set of vertices with one arbitrary vertex from each connected component in D_b. Then $|V_0| \leq C$. Any coverage tour must visit each vertex in V_0 because each vertex in V_0 lies on a required edge. For the graph G_0, let T^* be the optimal ATSP tour, and T be the ATSP tour returned by the $\alpha(C)$-approximation algorithm. Then $c(T) \leq \alpha(C)\ c(T^*) \leq \alpha(C)\ c(\tau^*)$.

Let τ be the final coverage tour obtained by adding the arcs from T to the digraph D_b and generating an Eulerian tour. Then $c(\tau) = c(D_b) + c(T) \leq \beta\ c(\tau^*) + \alpha(C)\ c(\tau^*) = (\alpha(C) + \beta)\ c(\tau^*)$. ∎

Corollary 1. *Combining Theorems 4 and 5, and noting that the number of connected components C is usually small in practice, as stated by [40], we observe:*

1. *The single robot line coverage problem has an $\alpha(C) + 2$ approximation factor. This also improves the previously best-known approximation result of $\alpha(C) + 3$ for the asymmetric rural postman problem given by [40].*
2. *If $C \in \mathcal{O}(\log n)$, an $\mathcal{O}(C^2 2^C)$ dynamic programming algorithm gives the optimal ATSP solution in polynomial time, giving a 3-approximation algorithm for the single robot line coverage problem.*

Remark 2. For the special case when we have two connected components, we do not have an ATSP tour. In such a scenario, we can duplicate one of the vertices $v \in V_0$ and add a zero cost edge from the duplicated vertex to v. The edge set E_0 will then be created on these three vertices.

The final coverage tour for the example graph, given in Figure 3, is shown in Figure 9. For this example, the cost of the tour obtained using the approximation algorithm is optimal.

4.4 | Improvements and extensions

We now provide two heuristics that improve the quality of the solutions generated by the algorithms discussed in the paper. We explore the use of the generalized traveling salesperson problem (GTSP) instead of the ATSP. We discuss the practical aspects of implementing the algorithms. Finally, we discuss the application of our algorithm to the line coverage problem with multiple robots.

Short-circuiting: Our first heuristic is to improve the final coverage tour by replacing a sequence of consecutive deadheading edges with the shortest path from the first vertex of the sequence to the last vertex of the sequence.

2-opt heuristic: The quality of the solution can be further improved by employing a simple 2-change local neighborhood search, also known as 2-opt, similar to that for the TSP, see, for example, [37]. Since it is an *anytime* heuristic, that is, it always maintains a feasible solution, we can ensure that the total number of constant-time local moves is restricted to n^3 to maintain the $\mathcal{O}(n^3)$ running time of the algorithm. We discuss the computational costs and the improvement in the solutions on a dataset of 50 road networks in Section 5.

GTSP based algorithm: In the algorithm for the case of general required graphs, an arbitrary vertex was selected for each connected component to create an auxiliary graph G_0 required as input to the ATSP algorithm. Since the choice of the vertices may affect the cost of the tour, an alternative technique is to formulate the problem as a generalized traveling salesperson problem (GTSP). Each connected component forms a cluster, and the vertices in the connected component form nodes in the cluster. The

cost between any pair of nodes corresponds to the shortest deadheading path between the nodes. The GTSP is then to compute a minimum cost tour such that at least one of the nodes in each cluster is visited. A GTSP instance can be solved by converting it to an instance of the ATSP with n vertices, where n is the total number of nodes in the GTSP instance, as given by [29]. Such a solution has an approximation factor of $\alpha(n)+2$, where $\alpha(n)$ is the approximation factor of an algorithm for the ATSP on a graph with n vertices. In principle, a GTSP based algorithm can provide better solutions as we no longer select an arbitrary vertex from each connected component. However, in practice, the GTSP based algorithm may require a longer computation time and, therefore, is not always suitable for robotics applications that require rapid computation of the coverage tours for the robots.

Practical considerations: The $(22+\epsilon)$-approximation algorithm for the ATSP given by Traub and Vygen [39] is not very practical for robotics applications because of its running time and challenging implementation. However, it is very relevant for providing a constant-factor approximation guarantee. As the number of connected components is usually very small, the dynamic programming based algorithm of Held and Karp [22] works well in practice. The algorithm runs in $\mathcal{O}(n^2 2^n)$, where n is the number of vertices. Techniques for bitwise operations by Knuth [26] are used to run through the 2^n combinations. We also use a state-of-the-art solver for ATSP from Helsgaun [23] for instances with a larger number of connected components. The results are discussed in Section 5.

Multiple robots: Our approximation algorithms have implications for the algorithms for arc routing problems with multiple robots. In the capacitated arc routing problem (CARP), the edges of the graph have a demand associated with them, and the robots have a capacity Q, see, for example, [11]. The task is to find a set of tours for a team of k robots such that the total demand for any of the robots does not exceed its capacity Q. The objective function is the total cost of all the tours for the robots. One of the strategies for the CARP and its variants is first to find a large tour ignoring the demand constraints by employing algorithms for the single robot problem. Then the tour is split into smaller components that respect the capacity constraints [43]. Thus, any improvements to the algorithms for the single robot version improve the quality of the solution for the version with multiple robots. A tour-splitting algorithm was given by van Bevern et al. [40] for the CARP on mixed and windy graphs. The algorithm has an approximation factor of $8\alpha(C+1)+27$ in general and an approximation factor of 35 when the number of connected components C is small, that is, $C \in \log(n)$. Our results immediately improve these approximation factors to $8\alpha(C+1)+19$ and 27, respectively.

5 | SIMULATIONS AND EXPERIMENTS

We now establish the efficiency and efficacy of the presented algorithms for the single robot line coverage problem through simulations and experiments.

The algorithms are implemented in C++ and executed on a desktop computer with an Intel Core i9-7980XE processor. We take advantage of the advances in linear programming solvers and use Gurobi [21] to obtain solutions rapidly for the minimum cost flow problem. Smaller instances of the ATSP are solved using the dynamic programming algorithm given by Held and Karp [22], while larger instances are solved using the LKH solver from Helsgaun [23]. The GLKH solver by Helsgaun [24] is used to solve the GTSP instances. The algorithm for the RPP on mixed and windy graphs by van Bevern et al. [40] is also adapted for the single robot line coverage problem for comparison with the algorithms presented in this paper. The short-circuiting based tour improvement routine is applied to the solutions generated from each algorithm as it replaces consecutive deadheadings with the shortest deadheading paths. The SRLC-ILP formulation is solved using Gurobi and executed on a cluster node with 48 cores. The solutions from the approximation algorithm developed in this paper are used to provide an initial solution to the ILP formulation, which helps in upper-bounding the branch-and-bound algorithms and obtaining solutions faster. Solving an ILP to obtain an optimal solution can take a long time; it took around 20 h for one of the instances with 635 vertices and 730 required edges.

5.1 | Simulation results on road networks

An important application of the single robot line coverage problem is the mapping, inspection, and surveillance of road networks. We generated a dataset[4] consisting of road networks from the 50 most populous cities in the world. These road networks differ considerably in structure from one another, and thus allow testing of the algorithms on a variety of graphs. The data was obtained from OpenStreetMap [31] by selecting a bounding polygon of 0.5 to 2.0 km^2 area using a web-based tool. The dataset consists of road networks with 75 to 635 vertices and 93 to 730 required edges. As UAVs can fly from one location to another, non-required edges are added between each pair of vertices, resulting in tens of thousands of non-required edges. In the case of no-fly zones, the corresponding non-required edges can be pruned in practice. The servicing and deadheading speeds are set to 7 $m \cdot s^{-1}$ and

[4]The dataset and a tool for extracting road network data are available at: https://github.com/UNCCharlotte-CS-Robotics/LineCoverage-dataset.

$10\,\mathrm{m{\cdot}s^{-1}}$, respectively. A wind of $2\,\mathrm{m{\cdot}s^{-1}}$ is simulated from the south-west direction, that is, $\pi/4$ radians from the horizontal axis. These parameters are set based on real-world experiments discussed in the following subsection.

Denote the speed of the UAV by v and the wind speed by w. For the traversal of an edge from a tail vertex t to a head vertex h, let the travel vector $\mathbf{t}$ denote the vector from t to 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:

$$v_{\text{eff}} = w\cos\phi + \sqrt{v^2 - w^2\sin^2\phi} \tag{29}$$

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

$$c(t,h) = \frac{\|\mathbf{t}\|_2}{v_{\text{eff}}} \tag{30}$$

Here, $\|\mathbf{t}\|_2$ is the Euclidean distance from t to h. The speed v of the UAV is set to the servicing or deadheading speed according to its travel mode. Note that the cost function is asymmetric due to wind.

We use the following notation for brevity:

1. $\beta 2$-ATSP: Algorithm SRLC-2Approx along with the dynamic programming algorithm for the ATSP.
2. $\beta 2$-GTSP: Algorithm SRLC-2Approx along with the GLKH solver for the generalized ATSP.
3. $\beta 2$-ATSP-2opt: Algorithm SRLC-2Approx along with the dynamic programming algorithm for the ATSP and 2-opt improvement heuristic.
4. $\beta 3$-ATSP: Algorithm given by van Bevern et al. [40] along with the dynamic programming algorithm for the ATSP.

Figure 10 shows four of the fifty road networks in the dataset, along with the coverage tours obtained from the ILP formulation and the $\beta 2$-ATSP-2opt algorithm presented in the paper.

A cost comparison of the solutions obtained from the $\beta 2$-ATSP, the $\beta 2$-ATSP-2opt, and the $\beta 3$-ATSP algorithms is shown in Figure 11. The y-axis shows the percentage difference in cost with respect to the optimal solution, that is, $\frac{c-c^*}{c^*} \times 100$, where c is the cost of the coverage tour given by the corresponding algorithm, and c^* is the optimal solution obtained using the ILP formulation. The solutions obtained by our final algorithm $\beta 2$-ATSP-2opt are within 10% of the optimal solution. Algorithm SRLC-2Approx with the DP algorithm for ATSP, denoted by $\beta 2$-ATSP, generally performs better than the algorithm given by van Bevern et al. [40], denoted by $\beta 3$-ATSP. There seems to be no perceptible trend in cost difference percentage with the increase in the instance size.

The computation times, shown in Figure 12, were obtained by averaging over 100 runs. The algorithm $\beta 2$-ATSP is comparable in running time to $\beta 3$-ATSP while providing better solutions, in general. For the $\beta 2$-ATSP-2opt algorithm, additional time is spent to run the 2-opt heuristic, which runs very fast for smaller instances and takes up to an additional 2 s for some of the larger instances. All 50 instances were solved within 3 s with a mean time of 0.83 s and a median time of 0.73 s, using the $\beta 2$-ATSP-2opt algorithm.

Road networks, for which the solution obtained by the SRLC-2Approx algorithm has more than one connected component, require the additional step of selecting the vertices to form the auxiliary graph G_0. We select the first indexed required vertex in each connected component to form the auxiliary graph G_0. Hence, the process is deterministic for a given input graph. In general, one could randomly select a required vertex from each connected component. Note that the approximation factor is not affected by the selection of the vertices in G_0, as shown in Theorem 5. The $\beta 2$-GTSP algorithm uses GTSP, and thus considers all vertices in each connected component. Figure 13 shows comparisons of costs and computation time for the $\beta 2$-ATSP and the $\beta 2$-GTSP algorithms. The comparison is performed for the instances with at least three connected components in the required graph. Using the GTSP gives better solutions in general, as the algorithm has the flexibility to select any vertex in each of the connected components. In contrast, an arbitrary vertex is selected for each connected component for the $\beta 2$-ATSP algorithm. In only one of the instances (Ahmedabad), the final coverage tour obtained using the DP algorithm for the ATSP resulted in slightly better results than that for using the GTSP, and this is because the ATSP based solution was more favorable for the short-circuiting routine and resulted in a better solution overall. The $\beta 2$-ATSP algorithm is computationally much more efficient.

The simulation results[5] indicate that all the ATSP based algorithms are sufficiently fast. With a small additional computational cost for the 2-opt heuristic, the $\beta 2$-ATSP-2opt algorithm computes high-quality solutions.

5.2 | Experiments with UAVs on road networks

We performed line coverage on two different portions of the UNC Charlotte road network using a DJI Phantom 4 quadrotor UAV. Figure 1 shows a portion of the road network, and Figure 14 shows a network of lanes on a set of parking lots. The experiments

[5]Results are available as supplementary material and also in the repository https://github.com/UNCCharlotte-CS-Robotics/LineCoverage-dataset.

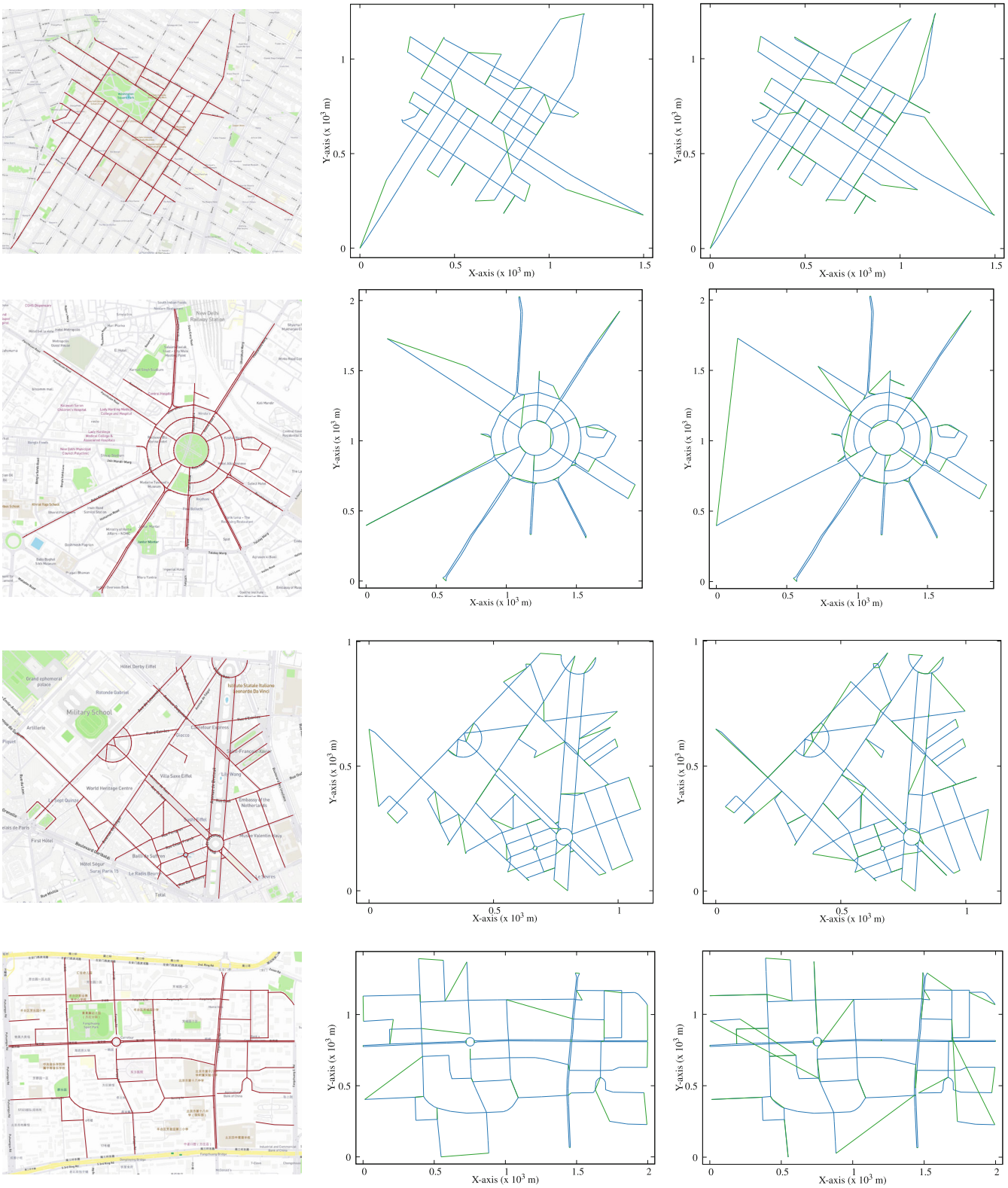


FIGURE 10 Four of the fifty sample road networks obtained from the most populous cities: The first column is the map representing the input graph, the second column is the optimal solution obtained using the SRLC-ILP formulation, and the third column is the final result of the algorithm (β2-ATSP-2opt) presented in this paper. The road networks, from top to bottom, are from (A) New York, (B) Delhi, (C) Paris, and (D) Beijing. Only the required edges representing the road network are shown on the map. There is a non-required edge for each pair of vertices in the graph. For example, the New York dataset has 1 connected component, 379 vertices, 402 required edges, and 71,631 non-required edges. The cost of the optimal coverage tour (middle column) generated by the ILP formulation is 2018.69, whereas the cost of the solution (right column) computed by the β2-ATSP-2opt algorithm is 2199.82. The solid blue lines represent servicing, while the dashed green lines represent deadheading travel.

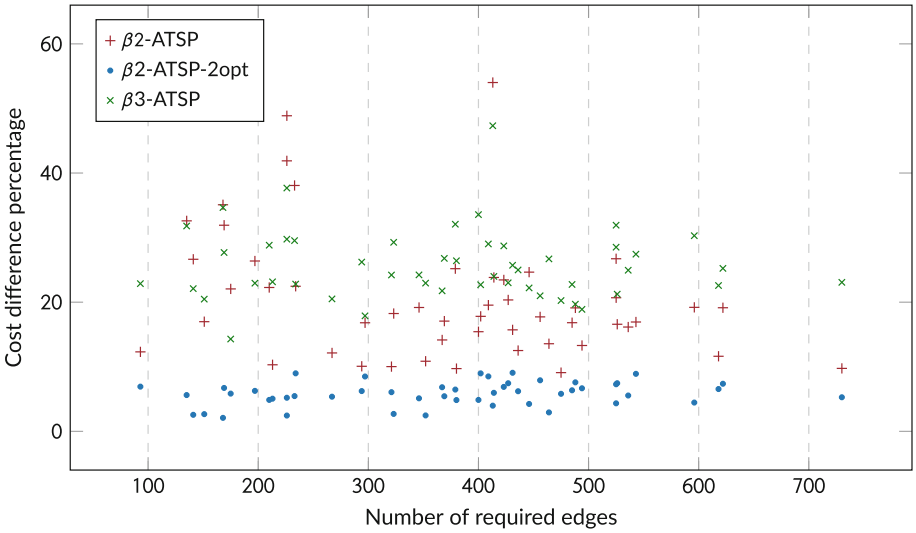


FIGURE 11 Cost comparison of the algorithms for the 50 road network dataset: The β2-ATSP algorithm, shown by red plus + markers, generally performs better than the β3-ATSP algorithm given by van Bevern et al. [40], shown by green cross markers ×. The solutions obtained by the β2-ATSP-2opt algorithm, shown by blue dots •, are consistently within 10% of the optimal.

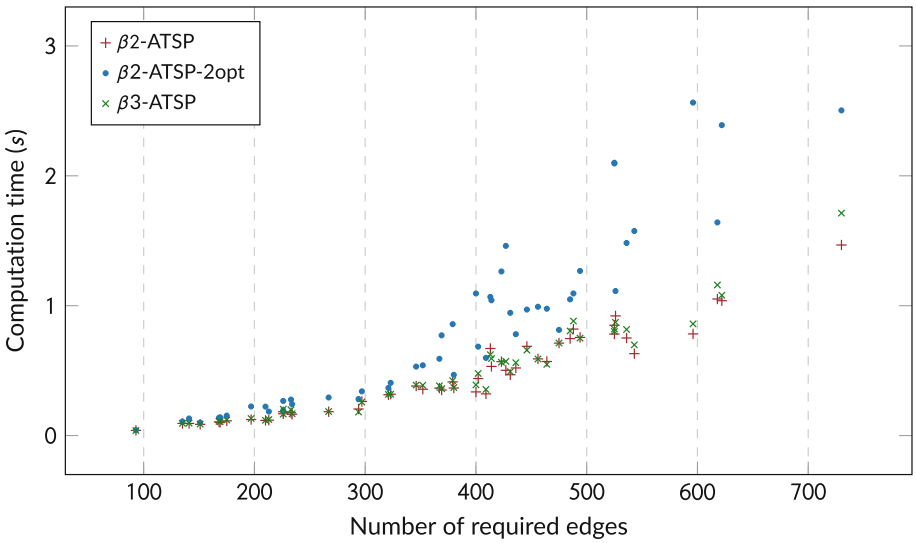


FIGURE 12 Computation time comparison of various algorithms in the paper: The β2-ATSP algorithm takes time comparable to the β3-ATSP algorithm. The 2-opt heuristic improvement over the β2-ATSP algorithm takes very small additional time for small instances and up to 2 s for larger instances. The running times are averaged over 100 runs.

were performed with two different sets of operating conditions. The servicing and deadheading speeds, along with wind speeds and directions, are specified in Table 2. The cost functions are based on the time to traverse the respective edge; the wind conditions make the costs asymmetric, as specified by Equations (29) and (30). The computed line coverage tour costs using the SRLC-ILP formulation and the β2-ATSP-2opts algorithm are provided in Table 2. The table also provides the actual flight times. Figures 15 and 16 show the computed coverage tours using the β2-ATSP-2opt algorithm, the actual flight paths, and orthomosaics for the two datasets. We generated orthomosaics from the images collected during the flights. The images are taken only during servicing (and not during deadheading), leading to a smaller number of images and reducing the time to compute the orthomosaic.

We have the following observations from our experiments.

1 The actual flight time differs from the computed flight time. Since we use a commercial mobile phone application to fly the UAV autonomously along the coverage tour, we do not have access to a model of the controller. As our formulation allows arbitrary cost functions, a high-fidelity model of the trajectory controller and wind effects can be incorporated for better results. Another aspect is that we do not model turning costs, and UAVs need to slow down to take sharper turns.

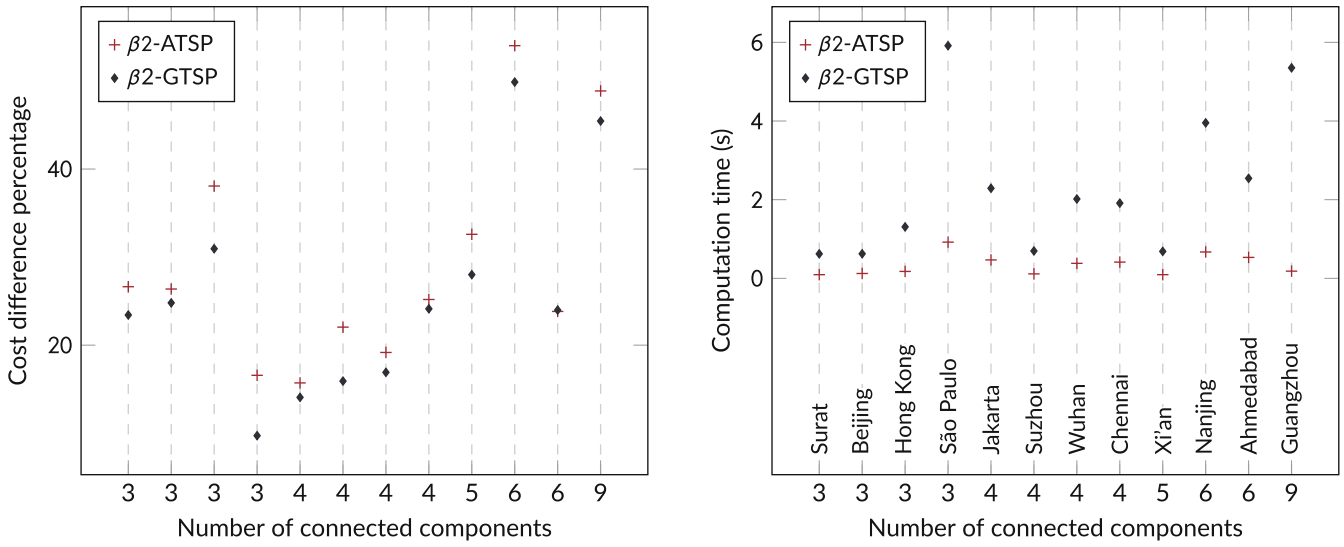


FIGURE 13 Computation time and cost comparisons for the $\beta 2$-ATSP and the $\beta 2$-GTSP algorithms: The $\beta 2$-GTSP algorithm gives better solutions, in general. However, the $\beta 2$-ATSP algorithm is computationally much more efficient. The comparison is performed for the instances with at least three connected components in the required graph.

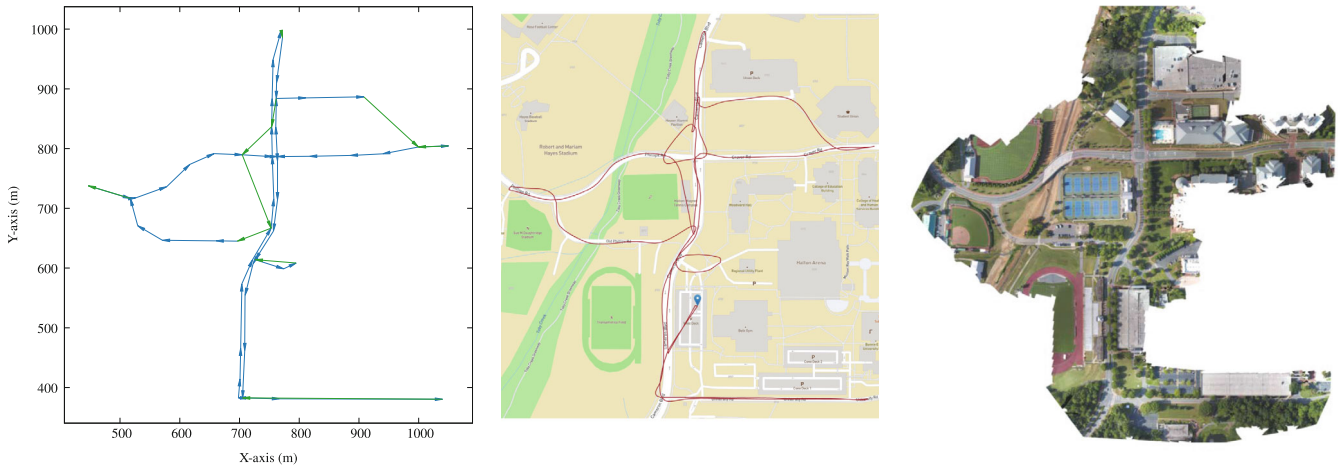


FIGURE 14 Coverage of a portion of the UNC Charlotte road network: The required graph has one connected component. The road network has a length of 2658 m with 48 vertices and 48 required edges. There are 1128 non-required edges formed by each pair of vertices. (A) Coverage tour generated using the $\beta 2$-ATSP-2opt algorithm. The cost of the solution is 492.48 s. The servicing travel is denoted by solid blue lines, while dashed green lines denote the deadheading travel. The arrowheads indicate the direction of travel. (B) The actual flight path of a UAV executing the coverage tour autonomously. The blue marker denotes the launch location of the UAV. (C) Orthomosaic generated from the images collected during servicing travel along the coverage tour. Collecting images only during servicing reduces the number of images that need to be processed for mapping and analysis.

TABLE 2 Operating conditions, computed coverage tour costs, and actual flight times for experiments with a quadrotor UAV.

	Road network	Parking lots
Service speed	7.00 $m \cdot s^{-1}$	3.33 $m \cdot s^{-1}$
Deadheading speed	10.00 $m \cdot s^{-1}$	5.00 $m \cdot s^{-1}$
Wind speed	2.00 $m \cdot s^{-1}$	1.34 $m \cdot s^{-1}$
Wind direction	45.00°	67.50°
SRLC-ILP cost	485 s	1123 s
SRLC-ILP flight time	502 s	925 s
$\beta 2$-ATSP-2opt cost	492 s	1172 s
$\beta 2$-ATSP-2opt flight time	527 s	1023 s

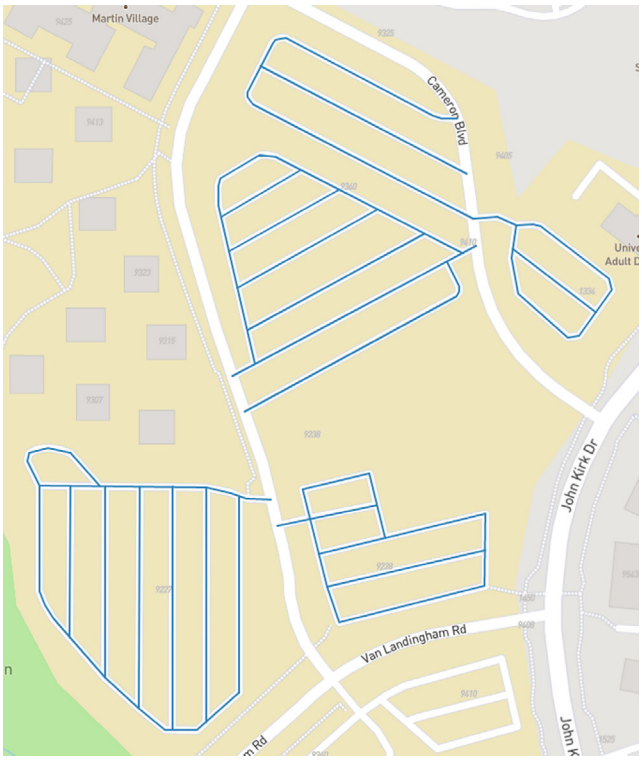


FIGURE 15 A network of lanes specified on a set of parking lots: The total length of the lanes is 2982 m. There are 90 vertices, 104 required edges, and 4005 non-required edges. The required edges form four connected components.

FIGURE 16 Line coverage of lanes specified on a set of parking lots: The required graph has four connected components. (A) Coverage tour generated using the β2-ATSP-2opt algorithm. The cost of the solution is 1172 s. The servicing travel is denoted by solid blue lines, while dashed green lines denote the deadheading travel. The arrowheads indicate the direction of travel. (B) The actual flight path of a UAV executing the coverage tour autonomously. The blue marker denotes the launch location of the UAV. The actual flight took 1023 s. (C) Orthomosaic computed from images collected during the flight.

This increases the actual flight time and indicates the importance of modeling the effect of turns in the objective function in the future.

2. Since we flew the UAV at a relatively high altitude (compared to the distance between parallel required edges and the sensor field of view), the generated orthomosaic provides an area coverage of the parking lots. Line coverage can, in fact, be used as a subroutine for area coverage [3].

In practice, UAVs are generally launched from an elevated position to maintain line of sight. Similarly, there may be additional physical and safety constraints that restrict the launch location of robots. Such a location need not be part of the road network, as shown by the blue marker in Figure 15B, and induce additional deadheading travel to and from the launch location. Although the cost of deadheadings from the launch location has not been considered in the experiments, one can easily incorporate them by adding an artificial required edge with zero service costs such that both the vertices of the edge correspond to the launch location. The two experiments demonstrate the use of our line coverage formulation and the algorithms to generate efficient coverage tours for linear infrastructure. The two modes of travel—servicing and deadheading—can be conveniently modeled in the formulation allowing lower operation times. Furthermore, allowing deadheading reduces the amount of sensor data required for analysis.

6 | CONCLUSION

Motivated by coverage applications for linear infrastructure such as road networks, power lines, and oil and gas pipelines, we addressed the single robot line coverage problem for autonomous aerial and ground robots. The linear features are modeled as required edges in a graph that the robot must service. Additional non-required edges, which do not require servicing, provide flexibility for a robot to select its path. The two modes of travel—servicing and deadheading—permit better modeling of real-world scenarios where a robot needs to perform task-specific actions such as taking images only along specified features. This reduces the workload of the robot, permits further optimization of the travel cost, and decreases the amount of sensor data that needs to be analyzed. Our formulation models asymmetric cost functions and permits multiple copies of edges. This enables one-way streets and repeated servicing of segments.

We formulated the single robot line coverage problem as an optimization problem on graphs and developed an ILP formulation that gives optimal solutions. Formal proofs establish the correctness of the formulation. As the problem is NP-hard, we developed approximation algorithms that provide a guarantee on the quality of the solutions. Studying the structure of the required graph—the graph induced by the linear features—provided insights into the problem, which were used to develop the approximation algorithms. The algorithms were developed in stages, going from a simple version of the problem to the most general one. First, an optimal algorithm based on the minimum cost flow problem was discussed for the case where the required graph is Eulerian. For the case where the required graph is connected but not necessarily Eulerian, a 2-approximation algorithm was developed. Finally, an $(\alpha(C) + 2)$-approximation algorithm was given for the general case of a required graph with C components, where $\alpha(C)$ is the approximation factor for an algorithm for the ATSP. Proofs for the approximation factor were provided for each of the algorithms. Heuristics that improve the quality of the solutions were incorporated into the algorithm, and a GTSP based alternative was evaluated.

Simulation results on a road network dataset of the 50 most populous cities in the world show that our main algorithm computes high-quality solutions that are within 10% of the optimum in less than 3 s. The algorithms are fast enough for rapid replanning. Experiments with a commercial UAV were performed on a portion of the UNC Charlotte road network and on lanes of a set of parking lots to generate orthomosaic maps. The autonomous flights resulted in fewer images that capture only the features of interest, as the images are taken only during servicing and not while deadheading.

We are currently exploring the application of our algorithms to the line coverage problem with multiple resource-constrained robots. Our plan is to use tour-splitting techniques to generate solutions for multiple robots. Our preliminary study indicates that very efficient solutions can be generated for the multi-robot line coverage problem using the high-quality solutions computed by our algorithms for the single robot line coverage problem.

ACKNOWLEDGMENTS

The map tiles in the figures use map data from Mapbox and OpenStreetMap and their data sources: https://www.mapbox.com/about/maps/ and http://www.openstreetmap.org/copyright. We thank Gurobi for making their ILP solvers available for academic use. We also thank Keld Helsgaun for releasing the programs LKH and GLKH for solving the ATSP and the GTSP.

DATA AVAILABILITY STATEMENT

The data that support the findings of this study are openly available in the LineCoverage-dataset repository at https://github.com/UNCCharlotte-CS-Robotics/LineCoverage-dataset.

ORCID

Saurav Agarwal https://orcid.org/0000-0003-1148-3186

REFERENCES

[1] S. Agarwal and S. Akella. *Line coverage with multiple robots*, IEEE International Conference on Robotics and Automation (ICRA), Paris, France. 2020, pp. 3248–3254.
[2] S. Agarwal and S. Akella, *Approximation algorithms for the single robot line coverage problem*, *Algorithmic foundations of robotics XIV*, S. M. LaValle, M. Lin, T. Ojala, D. Shell, and J. Yu (eds.), Springer International Publishing, Cham, Switzerland, 2021, pp. 534–550.
[3] S. Agarwal and S. Akella, *Area coverage with multiple capacity-constrained robots*, IEEE Robot Autom Lett **7** (2022), 3734–3741.
[4] E. M. Arkin, S. P. Fekete, and J. S. B. Mitchell, *Approximation algorithms for lawn mowing and milling*, Comput. Geom. **17** (2000), 25–50.
[5] R. Bellman, *Dynamic programming treatment of the travelling salesman problem*, J. ACM **9** (1962), 61–63.
[6] E. Benavent, A. Carrotta, A. Corberán, J. M. Sanchis, and D. Vigo, *Lower bounds and heuristics for the windy rural postman problem*, Eur. J. Oper. Res. **176** (2007), 855–869.
[7] E. Benavent, A. Corberán, E. Pinana, I. Plana, and J. M. Sanchis, *New heuristic algorithms for the windy rural postman problem*, Comput. Oper. Res. **32** (2005), 3111–3128.
[8] J. F. Campbell, A. Corberán, I. Plana, and J. M. Sanchis, *Drone arc routing problems*, Networks **72** (2018), 543–559.
[9] N. Christofides, *Worst-case analysis of a new heuristic for the travelling salesman problem*. Tech. report 388, Graduate School of Industrial Administration, Carnegie Mellon University, Pittsburgh, PA, 1976.
[10] A. Corberán, R. Eglese, G. Hasle, I. Plana, and J. M. Sanchis, *Arc routing problems: A review of the past, present, and future*, Networks **77** (2021), 88–115.
[11] A. Corberán and G. Laporte, Eds., *Arc routing: Problems, methods, and applications*, Society for Industrial and Applied Mathematics (SIAM), Philadelphia, PA, USA, 2014.
[12] A. Corberán and C. Prins, *Recent results on arc routing problems: An annotated bibliography*, Networks **56** (2010), 50–69.
[13] S. Dasgupta, C. Papadimitriou, and U. Vazirani, *Algorithms*, McGraw-Hill Higher Education, New York, USA, 2006.
[14] M. Dille and S. Singh. *Efficient aerial coverage search in road networks*, AIAA Guidance, Navigation, and Control Conference, Boston, MA, USA. 2013, pp. 5048–5067.
[15] K. Easton and J. Burdick. *A coverage algorithm for multi-robot boundary inspection*, IEEE International Conference on Robotics and Automation, Barcelona, Spain. 2005, pp. 727–734.
[16] J. Edmonds and E. L. Johnson, *Matching, Euler tours and the Chinese postman*, Math. Program. **5** (1973), 88–124.
[17] G. N. Frederickson, *Approximation algorithms for some postman problems*, J. ACM **26** (1979), 538–554.
[18] G. N. Frederickson, M. S. Hecht, and C. E. Kim. *Approximation algorithms for some routing problems*, 17th Annual Symposium on Foundations of Computer Science, Houston, USA. 1976, pp. 216–227.
[19] L. Gouveia, M. C. Mourão, and L. S. Pinto, *Lower bounds for the mixed capacitated arc routing problem*, Comput. Oper. Res. **37** (2010), 692–699.
[20] M. Guan, *On the windy postman problem*, Discr. Appl. Math. **9** (1984), 41–46.
[21] L. L. C. Gurobi Optimization. Gurobi optimizer reference manual. 2021.
[22] M. Held and R. M. Karp, *A dynamic programming approach to sequencing problems*, J. Soc. Indus. Appl. Math. **10** (1962), 196–210.
[23] K. Helsgaun, *An effective implementation of the Lin-Kernighan traveling salesman heuristic*, Eur. J. Oper. Res. **126** (2000), 106–130.
[24] K. Helsgaun, *Solving the equality generalized traveling salesman problem using the Lin-Kernighan-Helsgaun algorithm*, Math. Program. Comput **7** (2015), 269–287.
[25] N. Karapetyan, K. Benson, C. McKinney, P. Taslakian, and I. Rekleitis. *Efficient multi-robot coverage of a known environment*, IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), Vancouver, Canada. 2017, pp. 1846–1852.
[26] D. E. Knuth, *The art of computer programming, volume 4A: Combinatorial algorithms, part 1*, Addison-Wesley, Boston, MA, USA, 2011.
[27] J. K. Lenstra and A. H. G. R. Kan, *On general routing problems*, Networks **6** (1976), 273–280.
[28] R. Mannadiar and I. Rekleitis. *Optimal coverage of a known arbitrary environment*, IEEE International Conference on Robotics and Automation (ICRA), Anchorage, USA. 2010, pp. 5525–5530.
[29] C. E. Noon and J. C. Bean, *An efficient transformation of the generalized traveling salesman problem*, INFOR: Inform Syst. Oper. Res. **31** (1993), 39–44.
[30] H. Oh, S. Kim, A. Tsourdos, and B. White, *Coordinated road-network search route planning by a team of UAVs*, Int. J. Syst. Sci. **45** (2014), 825–840.
[31] OpenStreetMap contributors. Planet dump. 2022 https://planet.osm.org.
[32] J. B. Orlin, *A faster stronger polynomial minimum cost flow algorithm*, Oper. Res. **41** (1993), 338–350.
[33] C. S. Orloff, *A fundamental problem in vehicle routing*, Networks **4** (1974), 35–64.
[34] C. H. Papadimitriou and K. Steiglitz, *Combinatorial optimization: Algorithms and complexity*, Prentice-Hall, Inc., Hoboken, NJ, USA, 1982.
[35] B. Raghavachari and J. Veerasamy, *A 3/2-approximation algorithm for the mixed postman problem*, SIAM J. Discr. Math. **12** (1999), 425–433.
[36] B. Raghavachari and J. Veerasamy. *Approximation algorithms for the asymmetric postman problem*, Tenth Annual ACM-SIAM Symposium on Discrete Algorithms, Baltimore, Maryland, USA, Vol 1999, SODA. 1999, pp. 734–741.
[37] T. Roughgarden, *Algorithms illuminated, part 4: Algorithms for NP-hard problems*, Soundlikeyourself Publishing, LLC, New York, USA, 2020.
[38] O. Svensson, J. Tarnawski, and L. A. Végh. *A constant-factor approximation algorithm for the asymmetric traveling salesman problem*, 50th Annual ACM SIGACT Symposium on Theory of Computing, Los Angeles, CA, USA, Vol 2018, STOC. 2018, pp. 204–213.
[39] V. Traub and J. Vygen. *An improved approximation algorithm for ATSP*, 52nd Annual ACM SIGACT Symposium on Theory of Computing, Chicago, IL, USA, Vol 2020, STOC. 2020, pp. 1–13.
[40] 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 **70** (2017), 262–278.

[41] K. Williams and J. Burdick. *Multi-robot boundary coverage with plan revision*, IEEE International Conference on Robotics and Automation (ICRA), Orlando, USA. 2006, pp. 1716–1723.
[42] Z. Win, *On the windy postman problem on Eulerian graphs*, Math. Program. **44** (1989), 97–112.
[43] S. Wøhlk, *An approximation algorithm for the capacitated arc routing problem*, Open Oper. Res. J. **2** (2008), 8–12.
[44] L. Xu and A. Stentz. *An efficient algorithm for environmental coverage with multiple robots*, IEEE International Conference on Robotics and Automation (ICRA), Shanghai, China. 2011, pp. 4950–4955.
[45] L. Xu and T. Stentz. *A fast traversal heuristic and optimal algorithm for effective environmental coverage*, Proceedings of Robotics: Science and Systems, Zaragoza, Spain. 2010, pp. 161–168.

How to cite this article: S. Agarwal and S. Akella, *The single robot line coverage problem: Theory, algorithms, and experiments*, Networks. **82** (2023), 479–505. https://doi.org/10.1002/net.22171