IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Received 17 May 2024, accepted 7 June 2024, date of publication 13 June 2024, date of current version 26 June 2024.

Digital Object Identifier 10.1109/ACCESS.2024.3414331

RESEARCH ARTICLE

Hybrid Metaheuristic Approaches for the Multi-Depot Rural Postman Problem With Rechargeable and Reusable Vehicles

EASHWAR SATHYAMURTHY[1], JEFFREY W. HERRMANN[2], AND SHAPOUR AZARM[1]
[1]Department of Mechanical Engineering, University of Maryland, College Park, MD 20742, USA
[2]Department of Mechanical Engineering, The Catholic University of America, Washington, DC 20064, USA

Corresponding author: Eashwar Sathyamurthy (eashwar@umd.edu)

This work was supported in part by Maryland Industrial Partnerships (MIPS) Program at the University of Maryland, in part by SpringGem, in part by a gift from Dr. Alex Mehr (Ph.D. '03) through the Design Decision Support Laboratory Research and Education Fund, and in part by an Army Cooperative Agreement under Grant W911NF2120076.

ABSTRACT The limited battery life of unmanned autonomous vehicles and ground robots necessitates efficient routing strategies that enable periodic recharging to extend operational range. To address this issue, this paper studies an extension of the classical rural postman problem, where a fleet of rechargeable vehicles with limited battery capacity stationed at multiple depots must traverse required edges in a weighted graph through multiple trips, minimizing the total travel time. Existing heuristic approaches yield poor solutions with a high optimality gap due to the NP-hard nature of the problem. We propose a new Mixed Integer Linear Programming (MILP) formulation for the problem, which is used to obtain optimal vehicle routes and introduce new techniques to enhance existing metaheuristics, such as simulated annealing, tabu search, and genetic algorithm, to solve the problem. The routes obtained by testing the metaheuristics on benchmark and real-world instances based on roadmaps were compared to the optimal solution. Experimental results for benchmark instances showed that the simulated annealing and genetic algorithm kept the optimality gap within 1%. For real-world instances, simulated annealing and tabu search improved route quality by 54.1% and 49.9%, respectively, over routes obtained by only using heuristic approaches.

INDEX TERMS Multi-vehicle route planning, capacity constraints, metaheuristics, rural postman problem, rechargeable and reusable vehicles.

I. INTRODUCTION

The Multi-Depot Rural Postman Problem with Rechargeable and Reusable Vehicles (MD-RPP-RRV) addresses a critical real-world challenge - enabling efficient route planning for battery-operated unmanned aerial vehicles (UAVs) tasked with inspecting potentially icy road segments across a given area of interest (AOI). This problem is motivated by the need to utilize UAVs equipped with onboard sensors to inspect hazardous icy road segments, based on historical data analysis and expert insights, to warn drivers and prevent accidents [1].

The road network within the AOI is represented as a graph, with base stations forming a subset of nodes called depots and potentially icy road segments forming a subset of edges called required edges that need to be traversed. The research objective is to develop an optimized strategy for routing a fleet of UAVs with limited battery capacities to traverse these required edges while periodically returning to the depots for recharging. This ensures the UAVs maintain operational range, ensuring continued functionality during their inspection mission.

The associate editor coordinating the review of this manuscript and approving it for publication was Chun-Hao Chen.

The MD-RPP-RRV presents an arc routing problem [2] on an undirected weighted connected graph containing rechargeable vehicles with limited capacity stationed at multiple depots. The objective is to minimize the time needed to traverse the required edges of the graph at least once with a set of feasible routes for the vehicles. A vehicle can perform multiple trips; each trip begins at a depot, visits a sequence of nodes and possibly required edges, and ends at a depot, where the vehicle can recharge.

This paper formulates the MD-RPP-RRV into a MILP model and presents simulated annealing, tabu search, and genetic algorithm metaheuristics to improve the initial solution produced by the multi-trip algorithm (MT) heuristic [3]. The MD-RPP-RRV is a challenging combinatorial optimization problem that belongs to the class of NP-hard problems (Section II). Hence, the MT heuristic solutions exhibited poor quality with an average optimality gap of 162%, necessitating the use of metaheuristics to reduce this gap. Furthermore, the paper reports on a comparative analysis of performance for optimal solutions derived from MILP formulations utilizing the Gurobi optimizer [4] and the proposed integrated metaheuristic approaches.

The main contributions of this paper are the following:

1) A new class of arc routing problem called MD-RPP-RRV wherein rechargeable and reusable vehicles conduct multiple trips across various depots. While previous literature explores similar concepts in node routing problems [46], [47], [48], [49], [50] and multi-trip CARP [21], [32], this paper integrates rechargeability and reusability aspects for solving multi-depot arc routing problems with multiple trips.
2) The formulation of the problem as a MILP that can be used to find optimal routes.
3) Introduction of new ways to apply metaheuristics to solve the problem that notably enhances the heuristic solutions provided by the multi-trip algorithm. These metaheuristics demonstrate effectiveness across various instances, including those based on real-world roadmaps.
4) Experimental evaluation and comparison with optimal solutions generated by the Gurobi optimizer reveal that the simulated annealing algorithm is scalable and consistently produces reliable results for both benchmark and real-world instances.

The remainder of this paper is organized as follows: Section II presents a literature review of related works. Section III provides the assumptions considered and presents a MILP formulation for the MD-RPP-RRV. Section IV briefly describes the proposed metaheuristics. Section V provides the results obtained from testing the metaheuristics on instances for the MD-RPP-RRV. Section VI concludes the paper.

II. LITERATURE REVIEW

This section reviews the literature related to the MD-RPP-RRV. The MD-RPP-RRV is a variant of the Rural Postman Problem (RPP) [6], which is an arc routing problem that aims to find the minimum-length tour of a postman traversing a subset of arcs in a graph. Unlike the Chinese Postman Problem (CPP) that requires traversing all edges and can be solved in polynomial time [7], the RPP is proven to be NP-hard [8]. The MD-RPP-RRV inherits the NP-hardness of the RPP, as it reduces to the RPP when considering only a single depot and restricting vehicles to a single trip, forming a Hamiltonian cycle. The MD-RPP-RRV can represent many real-world problems, such as routing vehicles optimally for snowplowing or routing UAVs to inspect leaky gas lines or finding deformed railway tracks.

CPP and RPP are special cases of the well-studied Capacitated Arc Routing Problem (CARP). The CARP involves efficiently servicing a street network with capacity-constrained vehicles from a central depot, with the objective of minimizing total routing costs. While numerous strategies have been developed for the single-depot CARP variant [29], [30], [33], [35], [36], [37], research on the multi-depot CARP remains relatively scarce in the literature. The literature review focuses on the approaches developed for solving multi-depot versions of RPP and CARP, highlighting the differences from the studied problem.

The Multi-Depot CARP (MD-CARP) [22] is a variant of the classical CARP, where vehicles are stationed at multiple depots and must start and end their routes from those locations. The typical scenario involves vehicles returning to their respective depots, although variations exist where vehicles can return to any depot. MD-CARP often involves a heterogeneous fleet of vehicles and finds application in various fields, such as package delivery and waste management. Kansou and Yassine [24] proposed new upper bounds for the MD-CARP by developing a new memetic algorithm based on a special crossover and an Ant Colony Optimization (ACO) metaheuristic with an insertion heuristic. Hu et al. [23] proposed a Hybrid Genetic Algorithm with Perturbation (HGAP), which produced competitive solutions on MD-CARP benchmark instances compared to [24]. Yu et al. [25] provided several theoretical guarantees by proposing exact and constant-ratio approximation algorithms for MD-CARP.

Polacek et al. [33] explore a variant of the MD-CARP known as the CARP with Intermediate Facilities (CARP-IF). In this problem, there's one depot, but additional nodes called intermediate facilities (IF) are introduced. Vehicles start and end their routes at the depot but can recharge at any intermediate facility. These facilities can serve practical purposes like waste disposal sites or storage for winter gritting materials. The authors propose two lower bounds and two heuristics for CARP-IF. Another similar variant of CARP with mobile depots is the CARP with Refill Points (CARP-RP) which can be seen as a variation of the MD-CARP where the depots are travelling to the vehicle to recharge. In this setup, there are two types of vehicles: regular service vehicles that travel along edges to provide service and refill vehicles that travel to meet service vehicles anywhere

on the graph for refueling. Amaya et al. [34] presents a mathematical model for CARP-RP and proposes a Cutting Plane algorithm to solve the problem. In the MD-CARP and its variants, the vehicles only perform single trips from multiple depots, but in MD-RPP-RRV, vehicles are recharged and reused to perform multiple trips from multiple depots.

Multi-trip CARP, where each vehicle is capable of performing multiple trips, has also been studied in the literature but with single depots, not multiple depots. Tirkolaee et al. [32] created a Mixed-Integer Linear Programming model for the multi-trip CARP to minimize total costs for urban waste collection. Their model placed depots and disposal sites strategically within urban areas. To solve this, they introduced a hybrid algorithm using the Taguchi parameter design method with an Improved Max-Min Ant System (IMMAS). Their approach proved highly effective in solving both standard test problems and large-scale instances, showcasing its efficiency. An extension of this problem was studied by Tirkolaee et al. [21] proposing a Hybrid Genetic Algorithm for the multi-trip green CARP with the objective of minimizing the total route cost including the cost of generation and emission of greenhouse gases and the cost of vehicle usage. Multi-trip CARP and Multi-depot CARP have been separately studied in the literature but MD-RPP-RRV combines the multi-trip and multi-depot aspects of these two problems.

Compared to multi-trip and multi-depot variants of the CARP, Multi-Depot RPP (MD-RPP), has been less extensively studied in the literature. Fernández and Rodríguez-Pereira [20] studied the MD-RPP on undirected graphs and presented a Mixed-Integer Linear Programming formulation with two different objective functions: (i) minimize total routing costs and (ii) minimizing the length of the longest route, by proposing a branch-and-cut algorithm that solves 95% and 99% of the instances optimally respectively for these objective functions. Chen et al. [26] formulated the Min-Max MD-RPP (MMMDRPP) and developed an efficient tabu-search-based algorithm and proposed three novel lower bounds to evaluate the routes. The algorithm was tested to design routes for police patrolling in London, and the results demonstrate the efficiency of the algorithm in generating balanced routes. An important area for further investigation is the exploration of the Multi-Depot RPP (MD-RPP) in conjunction with multi-trip variants, a research gap that has yet to be fully addressed. MD-RPP-RRV aims to bridge this gap.

Due to the NP-hard nature of both the MD-CARP and the MD-RPP, most of the approaches proposed in the literature are metaheuristic and exact approaches. Since exact approaches require significant computational effort and their computational complexity increases exponentially, they are not suitable for solving larger instances. Hence, we choose to develop metaheuristics to solve the MD-RPP-RRV. As the MD-RPP-RRV differs from both the MD-RPP and MD-CARP, the metaheuristics developed in the literature cannot be directly used to solve the MD-RPP-RRV. Additional motivation to solve the MD-RPP-RRV can be attributed to the increasing prevalence of battery-operated unmanned vehicles, particularly in surveillance operations. Given their reliance on batteries, optimizing the routing of these vehicles is crucial, considering their limited battery life and the necessity for frequent recharging to ensure efficient task completion.

III. PROBLEM FORMULATION AND ASSUMPTIONS

This section outlines the key assumptions underlying the formulation of the optimization problem.

A. ASSUMPTIONS

For the MD-RPP-RRV, the following assumptions are made:

1) All vehicles start from their respective depots with full battery capacity.
2) The AOI is represented as a weighted connected graph.
3) Vehicles navigate within the AOI solely by traversing edges of the graph.
4) The depots at which vehicles start, stop, and recharge are located at nodes of the network (such as intersections of roads).
5) If an edge is required, at least one vehicle must traverse the edge.
6) All vehicles have the same capacity (operating time when fully charged).
7) All vehicles move at the same speed.
8) The time to recharge the vehicle battery is the same at all depots. Partial recharging is not possible.
9) The time required to traverse an edge depends only upon the length of the edge and the vehicle speed.

Note that the description in this paper uses the term ‘‘recharge’’ because our work was motivated by battery-powered UAVs. More generally, this also refers to refueling operations for vehicles that run on gasoline or other fuel, and thus the formulation is also valid for such vehicles.

B. PROBLEM FORMULATION

The motivation to formulate the MD-RPP-RRV comes from the increasing deployment of rechargeable autonomous vehicles like UAVs, ground robots, and electric vehicles in various applications, which necessitates optimized routing strategies that account for their limited battery capacities and recharging requirements. Failing to consider these constraints can lead to inefficiencies, higher costs, and mission failures. The MD-RPP-RRV formulation aims to minimize the total time required for a fleet of rechargeable vehicles to traverse all required edges while allowing periodic recharging at designated depots, adhering to battery constraints. Solving this MILP model enables optimal resource utilization, cost savings, and improved reliability in applications involving rechargeable autonomous vehicles for tasks such as monitoring, inspection, transportation, and delivery.

Let $G = (N, E, T)$ be the undirected weighted connected graph that models the road network in AOI where N is the set of nodes (road intersections), E is the set of edges ($(i, j) \in E$) that connect nodes ($i, j \in N$), and T is the set of edge weights ($t(i, j) \in T$). An edge represents a road segment; let $l(i, j)$ be the length of the road segment in meters. All vehicles traverse road segments at a constant speed S meters per second, and the edge weight $t(i, j)$ represents the time taken by a vehicle to traverse edge (i, j), calculated as $t(i, j) = \frac{l(i,j)}{S}$.

Let $E_u \subseteq E$ be the set of required edges that need to be visited. Let N_d be the set of depots where vehicles can start, stop, or recharge. $N_d \subseteq N$. Let C be the maximum time that a vehicle remains operational after charging; this is finite because a vehicle's battery has finite capacity. Let K be the number of vehicles. Let R_T be the time taken to recharge a vehicle. Let F be the maximum number of trips that a vehicle can make, which is determined using an iterative solution procedure (Section V-A1).

To formulate the MD-RPP-RRV as a MILP model, three sets of binary decision variables are introduced:

1) $x(k, f, i, j) = 1$ if vehicle k traverses edge (i, j) from node i to node j during its f-th trip, and 0 otherwise.
2) $y(k, f, d) = 1$ if vehicle k ends its f-th trip at depot node $d \in N_d$, and 0 otherwise.
3) $z(k, f) = 1$ if vehicle k uses its f-th trip, and 0 otherwise.

$B(k) \in N_d, \forall k = 1, \ldots, K$ is an array of constants representing the depot vehicle k is initially located at the start of the mission. The values of $B(k)$ are given. The objective function β represents the maximum total time needed by any vehicle to complete all its trips and recharge between trips. Let δ be a constant that is much greater than $|E|$, the number of edges.

In the MILP formulation given below, constraint (1) ensures that if vehicle k has initiated its first trip from depot node $B(k)$, it is tracked using $z(k, 1)$. Constraint (2) restricts unnecessary trips by ensuring that trip $f + 1$ is used only if trip f is used by vehicle k. Constraint (3) keeps track of the depot node $d \in N_d$ where each trip f of vehicle k has ended using $y(k, f, d)$. Constraint (4) forces vehicle k to start its trip f from the same depot node $d \in N_d$ it ended its previous trip $f - 1$ if necessary. Constraint (5) ensures that vehicle k, if it begins trip f, must end in any of the depot nodes $d \in N_d$.

Constraint (6) enforces an upper bound for maximum trip time for all vehicles considering recharge time. Note constraint (6) only adds recharge time for used trip $f > 1$ determined using the $z(k, f)$. Constraint (7) ensures that no trip is too long (exceeds C). Constraint (8) ensures that all trips made by each vehicle enter or leave depot nodes the same number of times. Constraint (9) makes sure all trips of each vehicle have a continuous flow. Constraint (10) ensures that all trips made by all vehicles traverse each required edge at least once.

Constraint (11) ensures that a vehicle traverses no edges during a trip that is not used. Constraint (12) is a subtour elimination constraint that is based on one used by Chen et al. [26]. The set S is a set of nodes that are not depots, $E(S)$ is the set of required edges between nodes in S, and $\delta(S)$ is the set of edges that have only one node in the set S. This constraint ensures that, if a vehicle traverses a required edge $(p, q) \in E(S)$, then it also traverses edges that enter and leave S. Unfortunately, the size of this MILP grows exponentially with the number of nodes. For constraint (12) alone, there are $2^{|N/N_d|}$ subsets S, so the formulation requires a total of $2^{|N/N_d|} \times K \times F$ copies of this constraint.

The following section describes the proposed metaheuristics to solve the MD-RPP-RRV.

IV. SOLUTION APPROACHES

This section describes a route design allocation heuristic that designs feasible routes to the MD-RPP-RRV and metaheuristics, simulated annealing, tabu search, and genetic algorithm by presenting their pseudocode and how it improves the initial solution constructed from the MT algorithm heuristic for the MD-RPP-RRV.

$$\min \ \beta$$

subject to:

$$\sum_{(B(k),j)\in E} x(k, 1, B(k), j) = z(k, 1), k = 1, \ldots, K \tag{1}$$

$$z(k,f) - z(k,f+1) \geq 0, \ k = 1, \ldots, K, f = 1, \ldots, F-1 \tag{2}$$

$$\sum_{\substack{(i,d)\in E,\\ d\in N_d}} x(k,f,i,d) = y(k,f,d), \ k = 1, \ldots, K, f = 1, \ldots, F \tag{3}$$

$$y(k,f-1,d) \geq \sum_{(d,j)\in E} x(k,f,d,j) \ k = 1, \ldots, K, \quad f = 2, \ldots, F, d \in N_d \tag{4}$$

$$z(k,f) - \sum_{d\in N_d} y(k,f,d) = 0, \ k = 1, \ldots, K, f = 1, \ldots, F \tag{5}$$

$$\sum_{f=1}^{F} \sum_{(i,j)\in E} x(k,f,i,j)t(i,j) + \Big(\sum_{f=1}^{F} z(k,f) - 1\Big) \times R_T \leq \beta, \quad k = 1, \ldots, K \tag{6}$$

$$\sum_{(i,j)\in E} x(k,f,i,j)t(i,j) \leq C, \ k = 1, \ldots, K, f = 1, \ldots, F \tag{7}$$

$$\sum_{\substack{(i,j)\in E,\\ i\in N_d}} x(k,f,i,j) - \sum_{\substack{(i,j)\in E,\\ j\in N_d}} x(k,f,i,j) = 0, \quad k = 1, \ldots, K, f = 1, \ldots, F \tag{8}$$

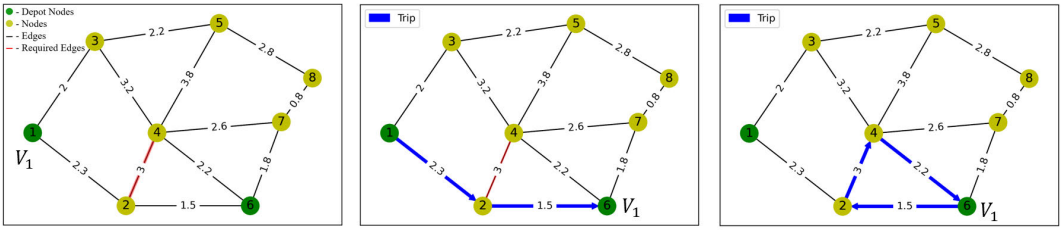

FIGURE 1. (a) A sample instance of MD-RPP-RRV, (b) Vehicle V_1 getting closer to the required edge. (c) Vehicle V_1 traversing the required edge.

$$\sum_{j\in N} x(k,f,i,j) - \sum_{j\in N} x(k,f,j,i) = 0,\ k = 1,\ldots,K,\ f = 1,\ldots,F, i \in N/\{N_d\} \tag{9}$$

$$\sum_{k=1}^{K}\sum_{f=1}^{F} x(k,f,i,j) + \sum_{k=1}^{K}\sum_{f=1}^{F} x(k,f,j,i) \geq 1,\ \forall (i,j) \in E_u \tag{10}$$

$$\sum_{(i,j)\in E} x(k,f,i,j) \leq z(k,f) \times M,\ k = 1,\ldots,K, f = 1,\ldots,F \tag{11}$$

$$\sum_{(i,j)\in\delta(S)} x(k,f,i,j) \geq 2 \times x(k,f,p,q),\ k = 1,\ldots,K,\ f = 1,\ldots,F, \forall S \subseteq N/\{N_d\}, (p,q) \in E(S) \tag{12}$$

$$x(k,f,i,j) \in [0,1],\ k = 1,\ldots,K, f = 1,\ldots,F, \forall (i,j) \in E \tag{13}$$

$$y(k,f,d) \in [0,1],\ k = 1,\ldots,K, f = 1,\ldots,F, d \in N_d \tag{14}$$

$$z(k,f) \in [0,1],\ k = 1,\ldots,K, f = 1,\ldots,F \tag{15}$$

$$\beta \in R^{+},\ \delta \gg |E| \tag{16}$$

A. MULTI-TRIP ROUTES

This section describes with an example a feasible solution to the MD-RPP-RRV to help the reader better understand the upcoming sections. A feasible solution specifies a route for each vehicle. A route includes one or more trips; each trip begins at a depot and ends at the same or different depot. The length of a trip is limited by the capacity constraint. Although its mission is to cover required edges, a vehicle might need to make a trip that includes no required edges in order to get closer to one or more required edges.

Consider an instance of the problem consisting of an undirected graph G with 8 nodes, 12 edges, 2 depot nodes, and 1 required edge, shown in Figure 1(a). There is only one vehicle (V_1), which begins at the depot at node 1; $C = 7$ time units; and $R_T = 1.1$ time units.

Due to the capacity constraint, the vehicle cannot directly traverse the required edge (2, 4) in one trip. The best possible trip to cover this edge is {1-2-4-6}, but this requires 7.5 time units, which exceeds vehicle capacity C, so it is infeasible.

TABLE 1. Nomenclature.

Symbols	Interpretation
$G = (N, E, T)$	Connected graph consisting of nodes, edges, edge weights
N_d	Set of depots $N \subseteq N_d$
E_u	Set of required edges
K	Number of vehicles
α	Set of current location of vehicles $\{\alpha_1, \alpha_2, .., \alpha_K\}$
R_T	Time to recharge a vehicle at a depot
C	Max time for a trip (capacity)
P	Set of routes for vehicles $\{P_1, P_2, .., P_K\}$
Y	Set of times that vehicles arrived at their last depot of their routes $\{Y_1, Y_2, .., Y_K\}$
Λ	Set of allocations of vehicle to s required edges $\{\Lambda_1, \Lambda_2, .., \Lambda_K\}$
M_i	Maximum number of iterations
N_g	Number of generations in Genetic Algorithm
P_s	Population size in Genetic Algorithm
p_c	Crossover probability
p_m	Mutation probability
p_r	Reproduction probability
p_e	Probability of accepting a better offspring

In a feasible multi-trip route, the vehicle travels from its current depot to another depot (node 6) to move closer to the required edge (Figure 1(b)), recharges at that depot, and then completes a trip that includes the required edge (Figure 1(c)).

The first trip {1-2-6} takes 3.8 time units for the vehicle to complete. Then, the vehicle recharges at node 6, which takes 1.1 time units. Finally, the vehicle takes the second trip {6-2-4-6}, which takes 6.7 time units to complete. Both trips are feasible because each one's duration is less than C. Completing the route and traversing the required edge requires 11.6 time units, which is the sum of both trip times and the recharge time and also the maximum trip time.

The following section describes a heuristic used by the metaheuristics to design routes for the MD-RPP-RRV, and Table 1 provides the nomenclature used in the following sections.

B. ROUTE DESIGN ALLOCATION ALGORITHM (RDA)

As defined earlier, the MD-RPP-RRV requires a subset of required edges E_u in the undirected weighted connected graph

Algorithm 1 Required Edges Traversal Based on Allocation

1: **procedure** RDA($G, \alpha, \Lambda, N_d, C, R_T$)
2: **for** k in 1 to K **do**
3: $E_t \leftarrow [\,],\ P_k \leftarrow [\,],\ A_k \leftarrow [\,]$
4: **while** Λ_k is not empty **do**
5: $T_p, \Lambda_k, E_t \leftarrow \text{trip}(G, \alpha_k, \Lambda_k, N_d, E_t, C)$
6: **if** time(T_p) $\neq 0$ **then**
7: $T_p, \Lambda_k, E_t \leftarrow \text{closedepot}(G, \alpha_k, \Lambda_k, N_d, E_t, C)$
8: **end if**
9: **if** E_t is not empty **then**
10: Add T_p to P_k
11: $\alpha_k \leftarrow$ end of last trip in T_p
12: Add E_t to A_k
13: $E_t \leftarrow [\,]$
14: **end if**
15: **end while**
16: Add P_k to P
17: Add A_k to A
18: **end for**
19: **return** P, A
20: **end procedure**

Algorithm 2 SA Algorithm

1: **procedure** SA($G, \alpha, P, N_d, C, R_T, T_{\max}, T_{\min}, C_r, \Lambda, M_i$)
2: $\Lambda_{\text{best}} \leftarrow \Lambda$
3: $P_{\text{best}} \leftarrow P$
4: $Y_{\text{best}} \leftarrow \text{triptimes}(P_{\text{best}}, C, R_T)$
5: $T \leftarrow T_{\max}$
6: **for** iter in 1 to M_i **do**
7: **if** $T > T_{\min}$ **then**
8: $Y \leftarrow \text{triptimes}(P, C, R_T)$
9: $\Lambda_{\text{new}} \leftarrow \text{newallocation}(\Lambda, Y, K)$
10: $P_{\text{new}}, \Lambda_{\text{new}} \leftarrow \text{RDA}(G, \alpha, \Lambda_n, N_d, C, R_T)$
11: $Y_{\text{new}} \leftarrow \text{triptimes}(P_{\text{new}}, C, R_T)$
12: $\Delta f \leftarrow \max(Y_{\text{new}}) - \max(Y)$
13: **if** $\Delta f < 0$ **then**
14: $\Lambda \leftarrow \Lambda_{\text{new}}$
15: $P \leftarrow P_{\text{new}}$
16: $Y \leftarrow Y_{\text{new}}$
17: **if** $\max(Y) < \max(P_{\text{best}})$ **then**
18: $P_{\text{best}} \leftarrow P$
19: $Y_{\text{best}} \leftarrow Y$
20: $\Lambda_{\text{best}} \leftarrow \Lambda$
21: **end if**
22: **else**
23: **if** $e^{(\frac{-\Delta f}{T})} > \text{random}(0, 1)$ **then**
24: $\Lambda \leftarrow \Lambda_{\text{new}}$
25: $P \leftarrow P_{\text{new}}$
26: **end if**
27: **end if**
28: **end if**
29: $T \leftarrow T \times C_r$
30: **end for**
31: **return** $P_{\text{best}}, \Lambda_{\text{best}}$
32: **end procedure**

G to be traversed at least once by the K rechargeable vehicles with limited capacity C from their respective depot nodes $N_d \subseteq N$ by minimizing maximum trip time. A feasible solution for the MD-RPP-RRV will have a set of routes P for K vehicles such that each vehicle route P_k for vehicle k will traverse a sequence (follows order) of the required edges $\Lambda_k \subseteq E_u$ such that $\cup_{i=1}^{K} \Lambda_i = E_u$. Let $\Lambda = \{\Lambda_1, \ldots, \Lambda_K\}$ represent the set of allocations of required edges for all vehicles. Hence, the problem of designing routes to solve the MD-RPP-RRV can be converted into an allocation problem of allocating E_u required edges among K vehicles such that routes resulting from the allocation minimize the maximum trip time the most.

However, determining maximum trip time requires knowing the routes of K vehicles. The allocation route design algorithm (Algorithm 1) takes in the allocated sequence of required edges Λ_k of each vehicle k as input and plans the routes of each vehicle k to traverse all the edges in Λ_k in order with one exception. The exception is if while trying to traverse a required edge $e_1 \in \Lambda_k$, which is at index 1 in Λ_k, the vehicle during its trip also traverses the required edge $e_3 \in \Lambda_k$, which is at index 3, the allocation algorithm will remove both the edges from Λ_k to avoid redundant traversal of required edges in subsequent iterations and also keeps track of actual sequence A_k in which the required edges were traversed. In A_k, which shows the actual sequence of traversal of required edges by vehicle k in routes P_k, the edges will be ordered $[e_1, e_3, \ldots]$. The edge e_3 will appear after e_1.

This allocation route design algorithm loops over each vehicle k with its respective allocation Λ_k (Lines 2-4) and tries to design trips to traverse the sequence of required edges in Λ_k in order using the trip subroutine (Line 5). The subroutine returns the routes for vehicle k consisting of trips T_p, each satisfying vehicle capacity C, the updated allocation Λ_k after removing the required edges traversed during the trips, and finally, the order in which the required edges were traversed E_t. T_p and E_t are used to update the routes of the vehicle P_k, the vehicle's current position α_k, and the actual allocation A_k if the trip T_p traversed any required edges in the sequence Λ_k (Lines 9-14). If vehicle k is not able to traverse a required edge in Λ_k, the closedepot() subroutine (Line 7) moves the vehicle to a depot closer to traverse required edge in Λ_k. The trip T_p used to move the vehicle k to a closer depot is also checked for required edge traversal in Λ_k (Line 9), as it is possible that vehicle k may traverse a future required edge in the sequence Λ_k. The pseudocode focuses on how, after allocating the required edges to the vehicles, the RDA algorithm will plan routes of vehicle k. The following section describes the simulated annealing metaheuristic, which uses the RDA and MT algorithm to design routes for the MD-RPP-RRV.

Algorithm 3 New Allocation Procedure

1: **procedure** NewAllocation(Λ, Y, K)
2: **if** random(0, 1) $>$ 0.5 **then** // INSERT
3: $k_{\max} \leftarrow \arg\max_k Y_k$
4: Remove last required edge $e_{\max}$ from $\Lambda[k_{\max}]$
5: Randomly select $k_{\text{insert}} \in \{1, 2, \ldots, K\} \setminus k_{\max}$
6: Insert $e_{\max}$ at the end of $\Lambda[k_{\text{insert}}]$
7: **else** // SWAP
8: Randomly select $k_1 \in \{1, 2, \ldots, K\}$
9: Randomly select $k_2 \in \{1, 2, \ldots, K\} \setminus k_1$
10: Randomly select edge e_1 from $\Lambda[k_1]$
11: Randomly select edge e_2 from $\Lambda[k_2]$
12: Swap e_1 and e_2 between $\Lambda[k_1]$ and $\Lambda[k_2]$
13: **end if**
14: **return** Updated allocation Λ
15: **end procedure**

C. SIMULATED ANNEALING (SA) ALGORITHM

The SA algorithm [52] takes a set of allocations (Λ) mapping vehicles to required edges and their associated routes (P) as input. The algorithm (Algorithm 2) seeks a better solution by reallocating the required edges among the vehicles.

A temperature parameter (T) is set (Line 5) to control the probability of accepting worse solutions during the optimization process. The algorithm then enters the main loop (Line 6), where it iteratively explores different allocations (Line 9) of required edges and associated routes of vehicles (Line 10) using the triptimes() subroutine and the RDA algorithm (Algorithm 1) respectively till the temperature parameter is greater than the minimum temperature ($T_{\min}$) (Line 7). The initial allocation Λ is either randomly generated or can be obtained using the MT algorithm by examining the routes.

The new allocation method (Algorithm 3) operates on an allocation Λ representing required edges assigned to a set of vehicles, a set containing trip times of all vehicle $Y = [Y_1, Y_2, \ldots, Y_K]$, and the total number of vehicles K. This procedure has two main operations (INSERT and SWAP), and it chooses one randomly; each has a probability of 0.5.

The INSERT operation (Lines 2-6 in Algorithm 3) identifies the vehicle with the maximum trip time and removes its last assigned required edge. Then, it randomly selects another vehicle and inserts the removed edge into its allocation.

The SWAP operation (Lines 7-12 in Algorithm 3) randomly selects two different vehicles, k_1 and k_2, and swaps a randomly chosen required edge from k_1 with an edge in k_2. The procedure returns the updated allocation Λ. This algorithm dynamically reallocates required edges among vehicles, which might improve the solution.

The SA algorithm (Algorithm 2) then updates the routes of vehicles that have new allocations (Line 10). Algorithm 1 is used to design the new routes (P_{K_n}), taking in the new allocation (Λ_n), and also updating the allocation based on the routes produced. If the new allocation produces routes with better maximum trip time (Line 11- 13), the new routes and allocation are updated with the current values (Lines 14-16), and the best routes and allocation area are also stored (Lines 18-20). However, if the new allocation results in a worse objective function, the algorithm will accept it with a probability that is determined by the Boltzmann criterion [52] (Lines 23-25). This probabilistic acceptance allows the algorithm to escape local optima and explore more of the search space.

As the algorithm proceeds, the temperature parameter decreases geometrically by the cooling rate (C_r) (Line 29), which reduces the probability of accepting worse solutions. This cooling schedule ensures that the algorithm eventually converges to a locally-optimal allocation. The SA algorithm finally returns the best allocation Λ_{best} and its associated vehicle routes P_{best} it converged to after the iterative process.

The following section describes the tabu search metaheuristic to solve the MD-RPP-RRV.

Algorithm 4 Tabu Search Algorithm

1: **procedure** TabuSearch($G, \alpha, E_u, N_d, C, \Lambda, R_T, M_i$)
2: $P, \Lambda \leftarrow$ RDA($G, \alpha, \Lambda, N_d, C, R_T$)
3: $P_{\text{best}} \leftarrow P$
4: $\Lambda_{\text{best}} \leftarrow \Lambda$
5: $t_l \leftarrow [\,]$
6: Add Λ to t_l
7: $Y \leftarrow$ triptimes(P, C, R_T)
8: $y_b \leftarrow \max(Y)$
9: **for** $it \leftarrow 1 to M_i$ **do**
10: $\Lambda_{\text{new}} \leftarrow$ newallocation(Λ, Y, K)
11: $P_{\text{new}}, \Lambda_{\text{new}} \leftarrow$ RDA($G, \alpha, \Lambda_{\text{new}}, N_d, C, R_T$)
12: **if** $\Lambda_{\text{new}} \notin t_l$ **then**
13: Add Λ_{new} to t_l
14: $Y_{\text{new}} \leftarrow$ triptimes(P_{new}, C, R_T)
15: $y_n \leftarrow \max(Y_{\text{new}})$
16: **if** $y_n < y_b$ **then**
17: $y_b \leftarrow y_n$
18: $P_{\text{best}} \leftarrow P_{\text{new}}$
19: $\Lambda_{\text{best}} \leftarrow \Lambda_{\text{new}}$
20: **end if**
21: **end if**
22: $\Lambda \leftarrow \Lambda_{\text{new}}$
23: $Y \leftarrow Y_{\text{new}}$
24: **end for**
25: **return** $P_{\text{best}}, \Lambda_{\text{best}}$
26: **end procedure**

D. TABU SEARCH (TS) ALGORITHM

TS is a powerful metaheuristic algorithm widely employed for solving combinatorial optimization problems. It operates by iteratively exploring the solution space while maintaining a short-term memory, known as the "tabu list," to guide the search process and prevent revisiting previously explored

solutions. This memory mechanism facilitates effective solution exploration and diversification, helping the algorithm avoid getting trapped in local optima.

The TS algorithm developed to solve the MD-RPP-RRV (Algorithm 4) takes in an allocation (Λ) as input and determines its corresponding routes P using algorithm 1 (Line 2). It then initializes variables to store the best routes and corresponding best allocation found so far, P_{best} and Λ_{best}, and adds the initial allocation to the empty tabu list t_l (Lines 3-6). It then computes the initial objective value y_b, which is the maximum trip time for the initial routes P (Line 8).

In the subsequent loop (Lines 9-21), the algorithm iterates through a fixed number of iterations M_i, generating new allocations of vehicles to required edges and updating solutions accordingly. Within each iteration, a new allocation Λ_{new} is computed (Line 10), followed by determining the corresponding routes P_{new}using the RDA algorithm (Line 11).

The algorithm evaluates the feasibility of the new allocation and computes its objective value (y_n) (Lines 15). If the new solution improves upon the current best solution, the algorithm updates the best solution, best allocation, and the best objective value (Lines 16-19). Finally, the algorithm returns the best set of routes P_{best} and the corresponding allocations Λ_{best} (Line 20). Through its systematic exploration and refinement process, TS efficiently navigates the solution space, aiming to minimize the objective function while adhering to problem constraints. The use of a tabu list prevents revisiting previously explored solutions, enhancing the algorithm's effectiveness in finding optimal or near-optimal solutions within a reasonable computational time.

The following section describes the genetic algorithm metaheuristic developed to solve the MD-RPP-RRV.

E. GENETIC ALGORITHM (GA)

This section describes the GA with the help of a pseudocode (Algorithm 5). It starts with an initial random allocation (Λ) of vehicles to required edges and tries to converge to an allocation (Λ_{best}) with better maximum trip time by iteratively improving the allocation with each generation using reproduction, crossover, and mutation operations. GA starts by creating a population (Lines 8 - 17), which is a set containing P_s number of uniquely random allocations λ_p along with their associated routes (ρ_p) and maximum trip times (γ_p) using RDA algorithm (Algorithm 1).

At the beginning of each generation, the GA makes a copy of the current population (Line 20) to decide if a particular member of the population needs to be replaced by new candidates at the end of the generation after undergoing reproduction, crossover, and mutation operations. In each generation, reproduction, crossover, and mutation operations are done probabilistically based on reproduction (p_r), crossover (p_c), and mutation (p_m) probabilities. After undergoing operations, the resulting population is evaluated

Algorithm 5 Genetic Algorithm

1: **procedure** GA($G, N_d, E_u, K, \alpha, R_T, C, N_g, P_s, p_c, p_m, p_r, p_e$)
2: $\Lambda \leftarrow$ Initial random allocation
3: $P, \Lambda \leftarrow$ RDA($G, \alpha, \Lambda, N_d, C, R_T$)
4: $P_{\text{best}} \leftarrow P$, $\Lambda_{\text{best}} \leftarrow \Lambda$
5: $y_{\text{best}} \leftarrow \max(\text{triptimes}(P_{\text{best}}, C, R_T))$
6: $\lambda_p \leftarrow [\,]$, $\rho_p \leftarrow [\,]$, $\gamma_p \leftarrow [\,]$
7: $g \leftarrow 0$, $i \leftarrow 0$
8: **while** $i < P_s$ **do**
9: $\Lambda_{\text{new}} \leftarrow$ newallocation(Λ, Y, K)
10: **if** $\Lambda_{\text{new}} \notin \lambda_p$ **then**
11: $P_{\text{new}}, \Lambda_{\text{new}} \leftarrow$ RDA($G, \alpha, \Lambda_{\text{new}}, N_d, C, R_T$)
12: Add Λ_{new} to λ_p
13: Add P_{new} to ρ_p
14: Add $\max(\text{triptimes}(P_{\text{new}}, C, R_T))$ to γ_p
15: $\Lambda \leftarrow \Lambda_{\text{new}}$
16: $i \leftarrow i + 1$
17: **end if**
18: **end while**
19: **while** $g < N_g$ **do**
20: $\lambda, \rho, \gamma \leftarrow \lambda_p, \rho_p, \gamma_p$
21: **if** random(0, 1) $\leq p_r$ **then**
22: $\lambda, \rho, \gamma \leftarrow$ reproduction(λ, ρ, γ)
23: **end if**
24: **if** random(0, 1) $\leq p_c$ **then**
25: $\lambda, \rho, \gamma \leftarrow$ crossover(λ, ρ, γ)
26: **end if**
27: **if** random(0, 1) $\leq p_m$ **then**
28: $\lambda, \rho, \gamma \leftarrow$ mutation(λ, ρ, γ)
29: **end if**
30: **for** $i \leftarrow 1 to P_s$ **do**
31: **if** $\min(\gamma_i) < \min(\gamma_{p_i})$ **then**
32: **if** random(0, 1) $\leq p_e$ **then**
33: $\gamma_{p_i} \leftarrow \gamma_i$, $\rho_{p_i} \leftarrow \rho_i$, $\lambda_{p_i} \leftarrow \lambda_i$
34: **end if**
35: **end if**
36: **end for**
37: $j \leftarrow \arg\min(\gamma_p)$
38: **if** $\gamma_{p_j} < y_{\text{best}}$ **then**
39: $y_{\text{best}} \leftarrow \gamma_{p_j}$, $\Lambda_{\text{best}} \leftarrow \lambda_{p_j}$, $P_{\text{best}} \leftarrow \rho_{p_j}$
40: **end if**
41: $g \leftarrow g + 1$
42: **end while**
43: **return** $P_{\text{best}}, \Lambda_{\text{best}}$
44: **end procedure**

(Lines 30 -36), and if they exhibit enhanced performance (Line 31), they may replace existing individuals in the population with a certain probability p_e (Lines 32-33). This follows the elitism concept [53] to retain some of the bad solutions along with the better solutions in the population to preserve diversity and escape local minima.

Algorithm 6 Reproduction Step

1: **procedure** reproduction(λ, ρ, γ, C, R_T)
2: $\lambda_{new} \leftarrow [\,]$, $\rho_{new} \leftarrow [\,]$, $\gamma_{new} \leftarrow [\,]$
3: $W \leftarrow [\,]$
4: $P \leftarrow [\,]$
5: **for** $i \leftarrow 1 to P_s$ **do**
6: $f_i \leftarrow \max(\text{triptimes}(\rho_i, C, R_T))$
7: Add $\frac{1}{f_i}$ to W
8: **end for**
9: **for** $i \leftarrow 1 to P_s$ **do**
10: Add $\frac{W_i}{\sum W}$ to P
11: **end for**
12: **for** $i \leftarrow 1 to P_s$ **do**
13: $j \leftarrow$ Randomly select index from λ based on W
14: $k \leftarrow$ Randomly select index from λ based on $W \setminus W_j$
15: **if** $W_j > W_k$ **then**
16: Add λ_j to λ_{new}
17: Add ρ_j to ρ_{new}
18: Add $\frac{1}{W_j}$ to γ_{new}
19: **else**
20: Add λ_k to λ_{new}
21: Add ρ_k to ρ_{new}
22: Add $\frac{1}{W_k}$ to γ_{new}
23: **end if**
24: **end for**
25: **return** λ_{new}, ρ_{new}, γ_{new}
26: **end procedure**

Algorithm 7 Crossover Step

1: **procedure** crossover($G, \alpha, N_d, C, R_T, \lambda, \rho, \gamma, K$)
2: $\lambda_{new} \leftarrow [\,]$, $\rho_{new} \leftarrow [\,]$, $\gamma_{new} \leftarrow [\,]$
3: **for** $i \leftarrow 1 to P_s$ **do**
4: $\Lambda_{new} \leftarrow \{\,\}$
5: Select λ_i from λ
6: Randomly select λ_j from $\lambda \setminus \lambda_i$
7: $Y_i \leftarrow \text{triptimes}(\rho_i, C, R_T)$
8: $Y_j \leftarrow \text{triptimes}(\rho_j, C, R_T)$
9: **for** $k \leftarrow 1 to K$ **do**
10: **if** $Y_i[k] < Y_j[k]$ **then**
11: $\Lambda_{new}[k] \leftarrow \lambda_i[k]$
12: **else**
13: $\Lambda_{new}[k] \leftarrow \lambda_j[k]$
14: **end if**
15: **end for**
16: $\Lambda_{new} \leftarrow \text{feasiblealloction}(\Lambda_{new})$
17: $P_{new}, \Lambda_{new} \leftarrow \text{RDA}(G, \alpha, \Lambda_{new}, N_d, C, R_T)$
18: Add Λ_{new} to λ_{new}
19: Add P_{new} to ρ_{new}
20: Add $\max(\text{triptimes}(P_{new}, C, R_T))$ to γ_{new}
21: **end for**
22: **return** λ_{new}, ρ_{new}, γ_{new}
23: **end procedure**

Algorithm 8 Mutation Step

1: **procedure** Mutation($G, \alpha, N_d, C, R_T, \lambda, \rho, \gamma, K$)
2: $\lambda_{new} \leftarrow [\,]$, $\rho_{new} \leftarrow [\,]$, $\gamma_{new} \leftarrow [\,]$
3: **for** $i \leftarrow 1 to K$ **do**
4: $Y \leftarrow \text{triptimes}(\rho_i, C, R_T)$
5: $\Lambda_{new} \leftarrow \text{NEWALLOCATION}(\lambda_i, Y, K)$
6: $P_{new}, \Lambda_{new} \leftarrow \text{RDA}(G, \alpha, \Lambda_{new}, N_d, C, R_T)$
7: Add Λ_{new} to λ_{new}
8: Add P_{new} to ρ_{new}
9: Add $\max(\text{triptimes}(P_{new}, C, R_T))$ to γ_{new}
10: **end for**
11: **return** λ_{new}, ρ_{new}, γ_{new}
12: **end procedure**

The reproduction step (Algorithm 6) involves making copies of individuals in the population with better fitness using tournament selection to create a new population (λ_{new}, ρ_{new}, γ_{new}) with better individuals. In tournament selection, tournaments are played by selecting two allocations (λ_j, λ_k) (Lines 13 -14) from the population based on their fitness, where the probability of selection is proportional to their fitness values (Lines 5 -11). The individuals with better fitness are placed in the mating pool (λ_{new}) (Lines 15 -22). As the selection of individuals is based on fitness, a fitter individual gets selected to participate in more tournaments, resulting in a mating pool with copies of individuals with better fitness and eliminating weaker individuals. The resulting mating pool is the updated population (Line 25) returned as a result of the reproduction step.

The crossover step (Algorithm 7) involves iterative selection of two allocations (λ_i, λ_j) (Lines 5-6) from the population and combining them to produce a better allocation (λ_{new}). This is done by computing the trip times (Y_i, Y_j) (Lines 7-8) for both allocations and comparing the trip times of each vehicle $k \in \{1, \ldots, K\}$. The trip time $Y_i[k]$ is the total time taken by a vehicle k takes to traverse all of its allocated required edges using single or multiple trips. The allocation with a better trip time out of λ_i and λ_j for vehicle k is added to a new allocation (Λ_{new}). Of course, the new allocation (Λ_{new}) may contain the same required edges allocated to multiple vehicles and, in the worst case, missing unallocated required edges. This is rectified using the *feasibleallocation*() subroutine (Line 16), which makes sure that the new allocation covers all the required edges. The crossover step is performed on the entire population, so in the end, a new population with better allocations is obtained.

The mutation step (Algorithm 8) makes use of the *newallocation*() (Algorithm 3) to alter each allocation in the population using INSERT and SWAP techniques to get a better allocation. After performing the reproduction, crossover, and mutation for N_g generation, the best routes P_{best} alongside its associated allocation Λ_{best} is returned. The GA explores a wider search space through reproduction, crossover, and mutation operations in each generation,

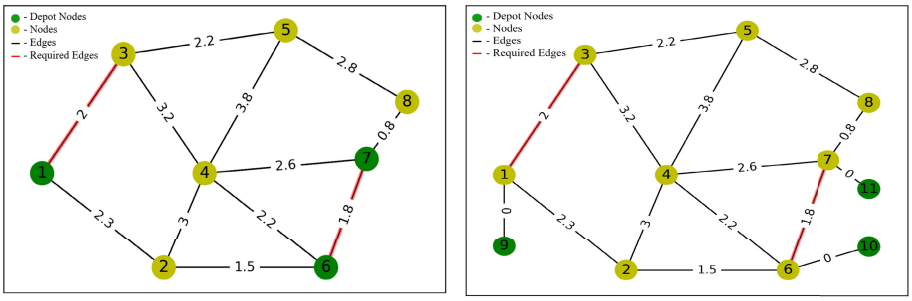

FIGURE 2. (a) A MD-RPP-RRV instance where required edges with one or both of its nodes as depot nodes, (b) Modified MD-RPP-RRV instance with dummy nodes and edges.

maintaining diversity. This approach may yield superior solutions compared to TS and SA, though it requires more computational effort. The following section discusses the results obtained from proposed metaheuristics on benchmark and real-world instances and compares those solutions with the optimal solutions obtained by solving the MILP.

V. RESULTS AND DISCUSSIONS

This section provides details about the instances that we used in our experiments and describes some important steps followed to obtain optimal results using the Gurobi optimizer version 10. Finally, the section discusses the results from the proposed metaheuristics, along with an analysis of the improvement in solution quality produced by each metaheuristic compared to the MT algorithm heuristic and the decrease in solution gap with respect to the optimum obtained by solving the MILP. The instances can be accessed from this GitHub repository link.

1) SET A INSTANCES

We created the Set A instances by converting 23 CARP instances with undirected graphs [17] to multi-depot capacitated RPP instances using the instance generation process. In these instances, C was set to equal twice the maximum edge weight of the instance. This was done so that a vehicle should be able to traverse the longest edge if it were chosen as one of the required edge.

2) SET B INSTANCES

Set B has eight instances created from real-world roadmaps of different regions with cold weather. We converted the roadmaps to undirected graphs and then to testing instances using the instance generation process. In these instances, C was set to 31 minutes, the listed capacity of the DJI Mavic Pro 2 UAV [54].

3) SET C INSTANCES

Set C has eight instances created from the Set B instances by adding the effect of wind on the edge travel times, which requires relaxing assumption 8 (Section III-A). In these instances, each edge had two weights (travel times): one for one direction and a second for the opposite direction. We determined the different weights by considering the impact of wind on a vehicle's travel time. In one direction, the wind may increase the vehicle's speed (relative to the ground), which reduces the travel time; in the other direction, the wind may decrease the speed and increase the travel time. Sathyamurthy [55] described the details of these calculations. For set C instances, as only the graph type is changed from undirected to directed, we decided to keep the number of vehicles (K), number of required edges (E_u), and the number of depot nodes (N_d), the same as set B instances. For required edges (E_u), only one direction of the edge, i.e., either (i, j) or (j, i), is randomly chosen to be the required edge as edge weights in either direction are different.

A. OPTIMAL SOLUTIONS USING GUROBI

This section describes the procedure used to determine the number of trips (F) needed by Gurobi to produce optimal solutions and also modifications done to the instances provided as input to the Gurobi optimizer.

1) DETERMINING NUMBER OF TRIPS (F)

From the MILP formulation (Section III-B), in order to define the decision variables programmatically in Gurobi, fixed values for the number of vehicles (K) and the number of trips (F) are needed. However, F cannot be known beforehand as each vehicle might take a different number of trips to traverse the required edges. Hence, F needs to be an upper bound of the number of trips of K vehicles. In order to find the upper bound, a simple iterative procedure is followed.

Initially, we set $F = 1$ and use Gurobi to find an optimal solution to the instance. If a feasible solution exists, then the instance is solved. If Gurobi reports that there are no feasible solutions, we increase F by 1 and attempt to solve it again. This continues until Gurobi finds a feasible, optimal solution.

The number of variables and the Random Access Memory (RAM) required to solve the problem increase as F increases. By starting with $F = 1$ and increasing it only as needed, this approach avoids solving any unnecessarily large instances, which is more efficient.

2) MODIFIED GUROBI OPTIMIZER INPUT

Because of the subtour elimination constraint (Constraint 12), if a vehicle traverses a required edge in the subset S, then it also traverses edges that enter and leave S. This is true for all possible subsets formed by the node set N/N_d. A problem occurs if a required edge has one or both of its nodes as depot nodes (Figure 2a). In this scenario, there are no subsets possible as subsets do not include depot nodes (N_d). To avoid this scenario, we added dummy nodes and edges with zero edge weights to modify any instance with such a required edge.

For example, in Figure 2b, the required edge $(1, 3)$ has node 1 as the depot node, and hence, a dummy node 9 and dummy edge $(1, 9)$ is inserted with an edge weight of 0, and the dummy node 9 is made the depot node instead of node 1. Note that the insertion of dummy nodes and edges has no effect on the quality of routes produced. A similar example is shown in Figure 2b with the required edge $(6, 7)$ where both of its nodes are depot nodes. Hence, two dummy nodes and edges are inserted and made depot nodes. The idea here is to isolate depot nodes from the instance so that it does violate any constraints of MILP formulation (Section III) thus restricting the search space of the Gurobi optimizer.

B. RESULTS

This section presents the results from our tests of the proposed metaheuristics and solving the MILP using Gurobi version 10.03 on the instances in Sets A, B, and C. These tests were run on an AMD EPYC 7763 64-core Processor with 128 physical cores, 128 logical processors, and 8 CPU cores. Up to 32 threads were used, and 8 GB of memory was allocated to each CPU core.

1) PARAMETER SETTINGS

The parameters used in the proposed metaheuristics were configured empirically from numerous tests performed with the benchmark and real-world instances. Table 2 presents the parameters used to test the metaheuristics on benchmark and real-world instances and their respective values.

TABLE 2. Parmeter setting.

Parameter	Value	Description
$T_{\max}$	100	Maximum temperature used in Simulated Annealing
$T_{\min}$	0	Minimum temperature used in Simulated Annealing
C_r	0.99	Temperature cooling rate
M_i	500	Number of iterations
N_g	50	Number of generation in GA
P_s	25	Population size in each generation in GA
p_r	0.1	Reproduction probability in each generation of GA
p_m	0.7	Mutation probability in each generation of GA
p_c	0.9	Crossover probability in each generation of GA
p_e	0.8	Probability of accepting a better offspring in each generation of GA

2) BENCHMARK INSTANCES RESULTS

Table 3 describes the set A instances, and tables 4 and 5 provide the results obtained by testing these instances on the proposed metaheuristics.

Table 4 shows the results obtained by metaheuristics SA and TS for 500 iterations, starting with the initial solution produced by the MT algorithm. Table 5 shows the results obtained by metaheuristics SA and TS for 500 iterations and GA.

The optimal objective function value, which is the optimal maximum trip time (M_{opt}) obtained from solving MILP using Gurobi for 19/23 set A instances, is also provided. For set A instances A.20 - A.23, the Gurobi optimizer could not produce optimal solutions using the MILP formulation as it ran out of allocated memory after running for more than 30 hours. For these instances, the results are represented as * in the table. Each set A instance is run 10 times, and the average (M), the best (best), and the standard deviation (STDEV) of maximum trip time along with the average execution time (ET) in seconds are provided in tables 4 and 5 to capture the stochastic nature of the metaheuristics.

To measure the quality of the metaheuristics solutions relative to the quality of the optimal solutions found by solving the MILP using Gurobi, we calculated the relative difference as follows. Let M be the average maximum trip

TABLE 3. Set A instances information.

Instance Name	N	E	E_u	C	K	N_d
A.1	8	11	3	18	1	2
A.2	11	19	5	40	2	3
A.3	7	21	8	16	4	2
A.4	7	21	8	12	4	2
A.5	12	22	7	40	3	3
A.6	11	22	8	18	4	3
A.7	12	22	8	40	4	3
A.8	12	22	7	44	3	3
A.9	12	22	8	40	4	3
A.10	13	23	7	60	3	3
A.11	12	25	9	38	4	3
A.12	12	26	9	40	4	3
A.13	13	26	7	44	3	3
A.14	10	28	9	198	4	3
A.15	8	28	10	16	5	2
A.16	8	28	10	14	5	2
A.17	11	33	11	18	5	3
A.18	9	36	13	16	6	2
A.19	11	44	14	18	7	3
A.20	22	45	16	38	8	5
A.21	27	46	13	18	6	6
A.22	27	51	16	18	8	6
A.23	11	55	18	16	9	3

TABLE 4. Benchmark Instances Results of SA and TS using MT heuristic. Instances where the metaheuristic converged to the optimum are denoted by bold highlighting in both the "Gap" and "B Gap" columns.

Instance		MT	MT+SA-500						MT+TS-500					
Name	M_{opt}	M_{MT}	M	STDEV	best	Gap	B Gap	ET	M	STDEV	best	Gap	B Gap	ET
A.1	64	64	64	0	64	**0.0**	**0.0**	0.0	64	0	64	**0.0**	**0.0**	0.0
A.2	148	244	148	0	148	**0.0**	**0.0**	3.7	230	0	230	55.4	55.4	3.5
A.3	15	15	15	0	15	**0.0**	**0.0**	3.2	15	0	15	**0.0**	**0.0**	3.4
A.4	7	32	7.2	0.4	7	2.9	**0.0**	3.1	8.2	1	7	17.1	**0.0**	3.0
A.5	126	140	127.4	1.7	126	1.1	**0.0**	3.3	130	0	130	3.2	3.2	3.2
A.6	16	58	17	0.8	16	6.3	**0.0**	3.9	16.8	1	16	5.0	**0.0**	3.9
A.7	34	138	34.4	0.5	34	1.2	**0.0**	2.9	34	0	34	**0.0**	**0.0**	2.7
A.8	41	163	51.2	28.6	41	24.9	**0.0**	3.3	100.4	47.7	42	144.9	2.4	3.8
A.9	134	149	135	1.3	134	0.7	**0.0**	3.0	136	0	136	1.5	1.5	2.6
A.10	59	351	101.7	65	59	72.4	**0.0**	3.7	86.8	55.6	59	47.1	**0.0**	3.4
A.11	120	127	123.2	2.9	120	2.7	**0.0**	4.1	127	0	127	5.8	5.8	4.8
A.12	40	149	116.8	25.8	40	192.0	**0.0**	4.3	133.3	2.1	127	233.3	217.5	4.3
A.13	141	166	142.1	1.3	141	0.8	**0.0**	4.8	148.8	4.4	141	5.5	**0.0**	4.9
A.14	48	63	48	0	48	**0.0**	**0.0**	5.5	53.2	1.6	50	10.8	4.2	5.6
A.15	12	16	12.4	0.5	12	3.3	**0.0**	4.6	12.2	0.6	12	1.7	**0.0**	4.4
A.16	10	40	10	0	10	**0.0**	**0.0**	4.3	10	0	10	**0.0**	**0.0**	4.1
A.17	15	51	15	0	15	**0.0**	**0.0**	5.5	15.2	0.6	15	1.3	**0.0**	5.5
A.18	13	16	14.2	0.4	14	9.2	7.7	5.8	13.7	0.8	13	5.4	**0.0**	5.7
A.19	11	50	12.6	0.5	12	14.5	9.1	6.5	13.3	0.9	13	20.9	18.2	6.4
A.20	*	*	33.1	1.1	31	*	*	6.7	34	0	34	*	*	7.4
A.21	*	*	57.9	13.4	18	*	*	10.0	48.4	15.2	18	*	*	9.2
A.22	*	*	63.1	1.9	60	*	*	10.7	68	0	68	*	*	10.0
A.23	*	*	13.7	0.5	13	*	*	8.4	15.1	0.7	14	*	*	8.1
Average Gap for 19 instances		**162.0**				**17.5**	**0.9**					**29.4**	**16.2**	

time of a solution obtained from metaheuristics (SA, TS or GA). Let M_{opt} be the maximum trip time of an optimal solution of the MILP. The metric Gap is the percentage relative difference between these values:

$$\text{Gap} = \frac{M - M_{\text{opt}}}{M_{\text{opt}}} \times 100 \tag{17}$$

The B Gap is the best gap obtained by taking the best value of the maximum trip time obtained by the metaheuristics in 10 runs.

$$\text{B Gap} = \frac{\text{best} - M_{\text{opt}}}{M_{\text{opt}}} \times 100 \tag{18}$$

When relying solely on the MT algorithm, the average gap for these 19 instances stood at 162% [3]. However, employing metaheuristics made a significant difference. Metaheuristics MT+SA and MT+TS, starting with solutions generated by the MT algorithm (Table 4), led to better outcomes, reducing the average gaps to 17.5% and 29.4%, respectively, with the best average gaps further diminishing to 0.9% and 16.2%, respectively. While metaheuristics SA, TS, and GA exhibited similar patterns when commencing from a random solution, the average percentage gaps for SA and TS rose to 23.2% and 75%, respectively, compared to using solutions from the MT algorithm as starting points. The same could be said about the best percentage gap for SA and TS, which are 11.9% and 31.2%, respectively. This suggests that providing the metaheuristics with a suboptimal heuristic solution (MT algorithm) instead of a randomly generated one as the starting point can, on average, lead them to converge to a better solution.

GA, on the other hand, could achieve a better average and best percentage gap, which are 13.7% and 0% compared to other metaheuristics. In fact, GA was able to converge to the known optimum for all 19 set A instances. However, this came at the cost of computational time, where GA took an average of 84.7 seconds to solve each set A instance, and all the other metaheuristics took an average of under 5 seconds. This represents a 1600% increase in computational effort for smaller set A instances with the number of edges ranging between 11 and 55. This shows that GA, however effective in converging to a better solution, the heavy reliance on computational resources makes it a less favorable option to solve larger instances.

The following sections provide the results of the metaheuristics for set B and set C instances.

3) REAL-WORLD INSTANCES RESULTS

Tables 6 describe the 16 real-world instances in Sets B and C, and tables 7 and 8 provide the results obtained by metaheuristics on those instances. Because these instances have many more nodes and edges than the benchmark instances, we were unable to obtain optimal solutions to the MILP using Gurobi due to the large compute and memory requirements.

Since the optimal maximum trip times of the set B and set C instances could not be determined, the solution quality produced by the metaheuristics was measured by considering the MT heuristics solution as the baseline. Let M_{MT} be the maximum trip produced by the MT algorithm heuristic and M and best be the average and best maximum trip produced

TABLE 5. Benchmark Instances Results of SA, TS and GA using randomly generated initial solutions. Instances where the metaheuristic converged to the optimum are denoted by bold highlighting in both the "Gap" and "B Gap" columns.

Instance		SA-500						TS-500						GA					
Name	M_{opt}	M_{MT}	STDEV	best	Gap	B Gap	ET	M	STDEV	best	Gap	B Gap	ET	M	STDEV	best	Gap	B Gap	ET
A.1	64	68	0	68	6.3	6.3	0.0	68	0	68	6.3	6.3	0.0	64	0	64	**0.0**	**0.0**	9.1
A.2	148	148	0	148	**0.0**	**0.0**	3.6	224.8	38.7	148	51.9	**0.0**	3.6	148	0	148	**0.0**	**0.0**	16.1
A.3	15	15.1	0.3	15	0.7	**0.0**	3.2	27.1	18.3	15	80.7	**0.0**	3.1	15	0	15	**0.0**	**0.0**	32.8
A.4	7	7	0	7	**0.0**	**0.0**	3.0	7.9	0.9	7	12.9	**0.0**	3.3	7	0	7	**0.0**	**0.0**	38.6
A.5	126	128	2.4	126	1.6	**0.0**	3.3	149	0	149	18.3	18.3	3.8	126	0	126	**0.0**	**0.0**	21.2
A.6	16	17	0.8	16	6.3	**0.0**	3.9	28.9	17.5	16	80.6	**0.0**	3.9	16.2	0.4	16	1.3	**0.0**	43.0
A.7	34	61.5	40.3	34	80.9	**0.0**	2.9	102.6	44.5	35	201.8	2.9	2.7	43.1	26.6	34	26.8	**0.0**	28.3
A.8	41	51.2	28.6	41	24.9	**0.0**	3.5	131.5	29.7	43	220.7	4.9	3.8	41.4	0.8	41	1.0	**0.0**	21.8
A.9	134	135.4	1.4	134	1.0	**0.0**	2.9	140.1	3.2	136	4.6	1.5	2.9	134.6	0.9	134	0.4	**0.0**	28.7
A.10	59	100.6	63.7	59	70.5	**0.0**	3.5	203	0	203	244.1	244.1	4.4	73.5	43.2	59	24.6	**0.0**	**20.0**
A.11	120	121	0.8	120	0.8	**0.0**	4.1	124.4	5.6	120	3.7	**0.0**	3.9	121.4	2	120	1.2	**0.0**	47.2
A.12	40	128	2.7	124	220.0	210.0	4.3	138.3	6.7	124	245.8	210.0	4.5	116.2	25.5	40	190.5	**0.0**	45.6
A.13	141	142.1	1.3	141	0.8	**0.0**	4.8	149.4	3	144	6.0	2.1	4.6	142.2	1.2	141	0.9	**0.0**	32.0
A.14	48	48.1	0.3	48	0.2	**0.0**	5.4	53.9	5.5	48	12.3	**0.0**	5.7	48	0	48	**0.0**	**0.0**	89.9
A.15	12	12.5	0.5	12	4.2	**0.0**	4.6	16	0	16	33.3	33.3	4.5	12.3	0.5	12	2.5	**0.0**	75.6
A.16	10	10	0	10	**0.0**	**0.0**	4.3	10.5	0.7	10	5.0	**0.0**	4.1	10	0	10	**0.0**	**0.0**	86.1
A.17	15	15	0	15	**0.0**	**0.0**	5.4	16	0	16	6.7	6.7	5.8	15	0	15	**0.0**	**0.0**	87.9
A.18	13	14	0.6	13	7.7	**0.0**	5.9	30.4	18.3	14	133.8	7.7	6.2	13.4	0.5	13	3.1	**0.0**	117.5
A.19	11	12.6	0.5	12	14.5	9.1	6.5	17.2	0.4	17	56.4	54.5	7.0	11.9	0.5	11	8.2	**0.0**	211.7
A.20	*	34.6	2.1	29	*	*	6.6	137	0	137	*	*	9.0	33	2	29	*	*	164.4
A.21	*	64.3	3.4	58	*	*	10.1	103.2	34.4	64	*	*	10.6	62.7	3.1	56	*	*	131.9
A.22	*	61.4	3.3	58	*	*	10.7	92.3	19.1	69	*	*	11.4	60.3	3.7	57	*	*	256.2
A.23	*	13.9	0.7	13	*	*	8.4	43.5	13.8	16	*	*	8.6	13.6	0.7	13	*	*	341.7
Average Gap for 19 instances					**23.2**	**11.9**					**75.0**	**31.2**					**13.7**	**0.0**	

TABLE 6. Set B and C instances information.

Instance Name	N	E	E_u	C	K	N_d
B.1	75	130	39	31	19	16
B.2	133	214	28	31	14	27
B.3	222	344	115	31	57	45
B.4	192	353	113	31	56	39
B.5	290	423	44	31	22	59
B.6	288	494	163	31	81	58
B.7	374	622	85	31	42	75
B.8	461	879	286	31	143	93
C.1	75	260	39	31	19	16
C.2	133	428	28	31	14	27
C.3	222	688	115	31	57	45
C.4	192	706	113	31	56	39
C.5	290	846	26	31	13	59
C.6	288	988	163	31	81	58
C.7	374	1244	81	31	40	75
C.8	461	1758	286	31	143	93

by the metaheuristics for set B and set C instances. Then, we define percentage improvement (% I) and best percentage (% B I) improvement (in 10 runs) of maximum trip time of the metaheuristics w.r.t to MT algorithm heuristics as follows:

$$\%\,\mathrm{I} = \frac{M_{MT} - M}{M_{MT}} \times 100 \tag{19}$$

$$\%\,\mathrm{B\ I} = \frac{M_{MT} - \text{best}}{M_{MT}} \times 100 \tag{20}$$

Table 7 showcases the results obtained from employing MT+SA and MT+TS, utilizing the MT algorithm's heuristic solution as their initial starting point. Notably, both SA and TS demonstrate considerable enhancements in optimizing the maximum trip time of instances in set B and set C. On average, MT+TS improves the maximum trip time by 49.9%, while MT+SA achieves a slightly higher improvement of 54.1%. However, delving deeper into the best percentage improvement reveals MT+TS outperforming MT+SA with a notable 88.5% improvement compared to MT+SA's 69.2%. Despite this, MT+TS exhibits more variability in solution quality, as evidenced by its higher average standard deviation across instances (28.3 for MT+TS compared to 12.8 for MT+SA). This variability suggests that while TS can achieve better improvements, its convergence may be less consistent than that of SA within the same 500 iterations.

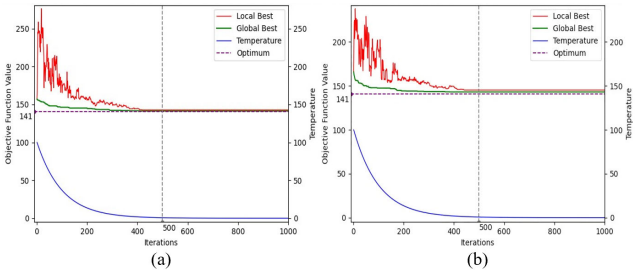

FIGURE 3. (a) Convergence plot of SA and (b) MT+SA for 1000 iterations, for instance, A.13.

Table 8 contains the results of SA, TS, and GA when initiated with a random feasible solution rather than leveraging the MT algorithm's heuristic. Interestingly, none of the metaheuristics manage to surpass the solutions obtained from the MT heuristic for all instances in set B and set C. With the exception of a few instances (B.2, B.5, C.2, and C.5), the metaheuristics predominantly yield negative values in both percentage improvement (% I) and percentage best improvement (% B I) columns.

TABLE 7. Set B and C Instances Results of SA and TS using MT heuristic.

Instance Name	MT	MT+SA-500						MT+TS-500					
	M_{MT}	M	STDEV	best	%I	%B I	ET	M	STDEV	best	%I	%B I	ET
B.1	28.7	28.7	0.1	28.6	0.0	0.3	51.5	28.7	0.0	28.7	0.0	0.0	46.1
B.2	356.7	275.0	48.5	199.0	29.7	79.2	111.3	228.6	28.1	203.5	56.0	75.3	97.1
B.3	30.8	23.9	1.2	21.7	28.9	41.9	195.1	15.8	1.5	14.0	94.9	120.0	199.8
B.4	127.2	30.6	0.0	30.6	315.7	315.7	271.5	37.3	0.0	37.3	241.0	241.0	190.1
B.5	809.5	333.7	43.0	262.1	142.6	208.9	489.4	552.3	171.5	272.5	46.6	197.1	385.9
B.6	130.3	129.5	1.1	127.8	0.6	2.0	650.8	106.2	6.0	103.1	22.7	26.4	513.0
B.7	343.0	298.8	33.4	256.2	14.8	33.9	889.1	224.5	24.5	200.2	52.8	71.3	737.4
B.8	122.6	122.6	0.0	122.6	0.0	0.0	1366.0	95.8	22.2	33.3	28.0	268.2	987.3
C.1	25.2	25.2	0.0	25.2	0.0	0.0	50.4	25.2	0.0	25.2	0.0	0.0	42.1
C.2	578.2	270.3	21.7	232.4	113.9	148.8	113.5	425.3	0.0	425.3	36.0	36.0	117.3
C.3	28.6	27.2	1.6	24.5	5.1	16.7	391.9	20.9	2.2	18.8	36.8	52.1	355.1
C.4	135.8	135.8	0.0	135.8	0.0	0.0	281.8	127.0	4.3	120.5	6.9	12.7	181.8
C.5	808.1	366.3	29.8	325.4	120.6	148.3	497.8	420.8	150.5	269.3	92.0	200.1	452.7
C.6	30.7	30.5	0.0	30.5	0.7	0.7	743.2	30.7	0.0	30.7	0.0	0.0	634.7
C.7	449.3	233.3	24.5	212.4	92.6	111.5	1435.0	267.8	34.5	244.5	67.8	83.8	856.8
C.8	122.2	122.2	0.0	122.2	0.0	0.0	1349.2	104.1	7.8	92.5	17.4	32.1	1043.2
Average % Improvement					**54.1**	**69.2**					**49.9**	**88.5**	

TABLE 8. Set B and C Instances Results of SA, TS and GA using randomly generated initial solutions. For GA, the results of the instances containing * are the best results obtained within 3600 seconds.

Instance Name	MT	SA-500						TS-500						GA					
	M_{MT}	M	STDEV	best	%I	%B I	ET	M	STDEV	best	%I	%B I	ET	M	STDEV	best	%I	%B I	ET
B.1	28.7	127.8	5.3	121.2	-77.5	-76.3	54.7	172.2	31.2	109.8	-83.3	-73.9	82.8	268.6	36.8	242.5	-89.3	-88.2	52.3
B.2	356.7	310.2	29.0	267.6	15.0	33.3	121.0	422.3	79.4	326.6	-15.5	9.2	142.7	623.8	110.4	525.5	-42.8	-32.1	69.7
B.3	30.8	47.8	36.8	27.9	-35.6	10.4	186.5	149.2	35.0	109.8	-79.4	-71.9	220.7	273.7*	42.1*	243.4*	-788.6*	-690.7*	3600
B.4	127.2	253.8	6.2	244.1	-49.9	-47.9	347.0	456.9	47.7	377.9	-72.2	-66.3	536.2	388.8	42.0	361.5	-67.3	-64.8	2810.0
B.5	809.5	338.6	34.5	292.8	139.1	176.5	487.0	394.7	58.9	324.3	105.1	149.6	579.1	615.0	93.4	550.5	31.6	47.0	158.5
B.6	130.3	201.1	40.6	149.2	-35.2	-12.7	840.7	474.6	22.4	456.0	-72.5	-71.4	1103.5	743.4*	119.9*	590.6*	-470.5*	-353.3*	3600
B.7	343.0	526.4	17.2	490.5	-34.8	-30.1	1112.0	802.7	31.2	773.9	-57.3	-55.7	1440.9	748.2	151.6	643.9	-54.2	-46.7	2144.0
B.8	122.6	383.0	7.6	373.2	-68.0	-67.1	2204.3	575.5	4.1	565.2	-78.7	-78.3	2888.8	1056.9*	84.4*	944.7*	-762.1*	-670.6*	3600
C.1	25.2	134.5	10.3	128.0	-81.3	-80.3	62.6	155.0	40.9	121.3	-83.7	-79.2	65.4	250.0	21.5	232.5	-89.9	-89.2	48.1
C.2	578.2	344.9	35.6	332.4	67.6	73.9	117.5	469.7	104.1	274.3	23.1	110.8	153.0	593.2	45.3	519.6	-2.5	11.3	92.0
C.3	28.6	31.0	3.5	29.6	-7.7	-3.4	178.6	179.1	21.6	115.5	-84.0	-75.2	773.2	349.9*	60.1*	251.3*	-1123.43*	-778.7*	3600
C.4	135.8	266.7	27.2	245.7	-49.1	-44.7	332.4	533.8	48.9	447.6	-74.6	-69.7	542.6	434.8	38.3	374.0	-68.8	-63.7	3358.6
C.5	808.1	478.9	74.5	441.9	68.7	82.9	535.3	480.6	133.5	343.6	68.1	135.2	610.2	646.6	50.7	564.5	25.0	43.2	257.9
C.6	30.7	251.7	23.9	239.9	-87.8	-87.2	916.2	347.2	22.3	288.7	-91.2	-89.4	1398.9	849*	153.1*	702.8*	-2665.8*	-2189.25*	3600
C.7	449.3	531.5	44.6	517.8	-15.5	-13.2	1112.4	678.9	56.8	635.7	-33.8	-29.3	1567.4	876.5*	34.6*	776.9*	-95.1*	-73*	3600
C.8	122.2	354.1	23.7	344.0	-65.5	-64.5	2220.3	809.5	59.6	736.7	-84.9	-83.4	3054.5	1129.7*	100.9*	980.8*	-824.5*	-702.6*	3600
Average % Improvement					**-19.8**	**-9.4**					**-44.7**	**-27.4**					**-443**	**-358.9**	

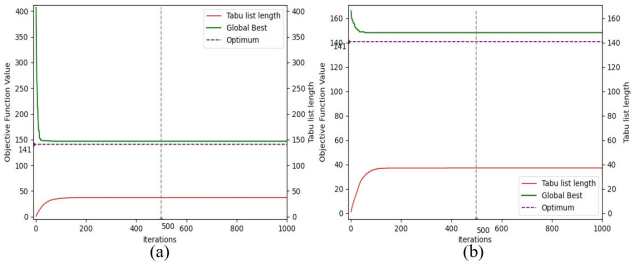

FIGURE 4. (a) Convergence plot of TS and (b) MT+TS for 1000 iterations, for instance, A.13.

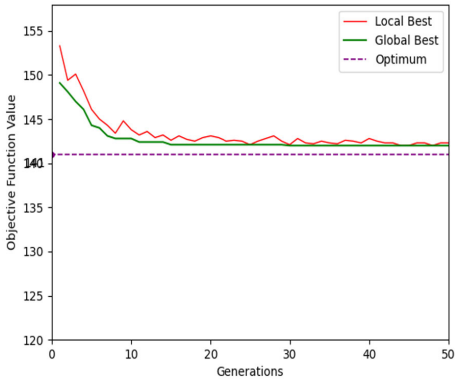

FIGURE 5. (a) Convergence plot of GA for 50 generations, for instance, A.13.

GA encounters additional challenges, failing to solve some instances altogether. To address this, a time limit of 3600 seconds was set for GA, representing the upper bound of computational time for all other metaheuristics. Instances where GA reached this time limit without converging are marked with an asterisk. The computational complexity of GA in a large practical real-world set B and set C instances is due to the generation of a unique initial population ($P_s = 25$), which is equivalent to solving the instances 25 separate times. On average, SA, TS, and GA demonstrate improvements in the solution quality for 4/16, 3/16, and 3/16 instances, respectively. This shows, for bigger instances, a limitation of metaheuristics in the form of dependence on a heuristic

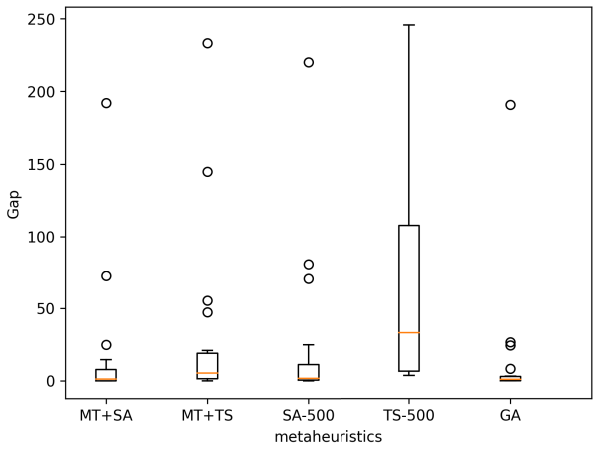

FIGURE 6. Box plot of average percentage gap (Gap) of metaheuristics for Set A instances.

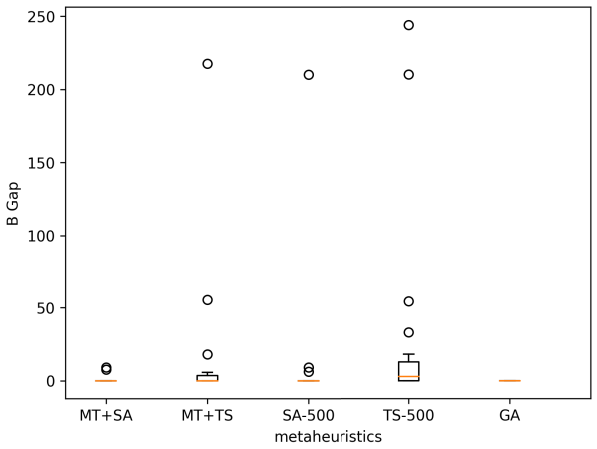

FIGURE 7. Box plot of best percentage gap (B Gap) of metaheuristics for Set A instances.

to give a good initial starting point to converge to a better solution.

4) ALGORITHM BEHAVIOR ANALYSIS

This section provides insights into the convergent behaviors of the proposed metaheuristics.

Figures 3a and 3b present the convergence graph with the averages of the temperature curves, the objective function for the current iteration (Local Best), and the objective function (maximum trip time) till the current iteration (Global Best) for 10 executions of set A instance A.13 SA starting with random initial and MT (MT+SA) solutions, respectively. From both these graphs, during initial iterations, there's a significant variation in the objective function value for the best local solution. This variation is mainly because of the higher temperature, which allows for the acceptance of poorer solutions (diversity), promoting exploration across the search space. As the temperature decreases, there's a decline in the acceptance of poorer solutions, indicating that optimization is narrowing down to a specific region and converging to the optimal solution.

Figures 4a and 4b present the convergence graph with the averages of the tabu list lengths and the objective function till current iteration (Global Best) for 10 executions of an example instance (A.13) for tabu search starting with random initial (TS) and MT solutions (MT+TS), respectively.

Initially, the length of the tabu list increases, indicating exploration of the search space for new solutions. Eventually, this growth subsides, signaling convergence, where no new solution is found, and the best solution obtained is returned. An important distinction arises in figure (4a), where the starting objective function value of the random initial solution is 401, significantly distant from the optimal value of 141. This underscores the importance of using a heuristic solution (MT) as the starting point of the metaheuristic, as depicted in figure (4b), where the heuristic solution yields an objective function value of 162 as the starting point of tabu search.

Figures 5 present the convergence graph with averages of the objective function for the current generation (Local Best), and the objective function (maximum trip time) till the current generation (Global Best) for 10 executions of set A instance A.13 for GA. The mutation (p_m), reproduction (p_r) and crossover (p_c) rate (Table 2) are specifically chosen after trial and error in a way to aggressively explore the search space to converge to a better solution with 50 generations. While the convergence graph shares similarities with those of other metaheuristics, GA's distinctive characteristic lies in its modification of 25 solutions (population) in each generation. The local best corresponds to the minimum objective function value among these 25 solutions in each generation. Consequently, the fluctuations observed in the local best reflect the continuous exploration of the solution space by GA's diverse population.

For numerous instances across sets A, B, and C, the solutions generated by all metaheuristics showed no improvement after 500 iterations. Consequently, we set the number of iterations (M_i) to 500 and the number of generations for GA to 50 (N_g). This decision was guided by the aim to strike a balance between achieving effective optimization and minimizing computational effort, particularly for larger instances within Sets B and C.

5) DISCUSSION

This section discusses some important observations noticed during the experimentation and provides insights.

In set A, finding high-quality solutions for instance A.12 was a challenge for many of the metaheuristics (except GA and MT+SA), which had a best gap of over 200% for this instance. Their poor performance on this instance significantly affected their average performance, as shown in Figures 6 and 7, which show the average (Gap) and best percentage (B Gap) gaps for set A instances for all the metaheuristics, respectively.

We can see the effect of instance A.12, which resulted in the maximum average and best percentage gaps for all

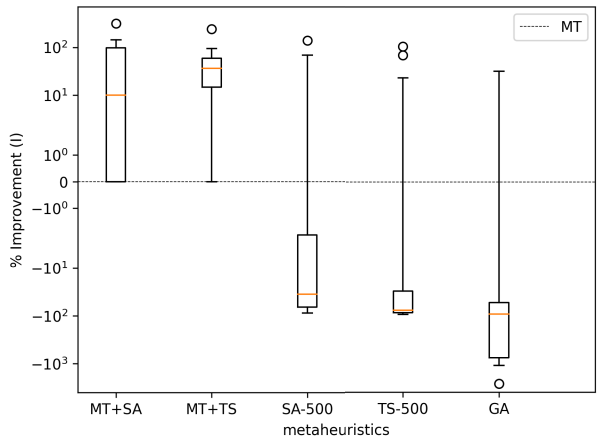

FIGURE 8. Box plot of average percentage improvement (% I) in solution quality of metaheuristics for Set B and C instances.

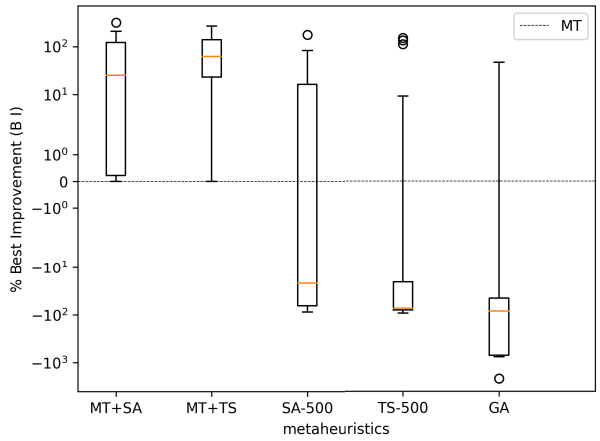

FIGURE 9. Box plot of best percentage improvement (%B I) of metaheuristics for Set B and C instances.

metaheuristics, causing variability in the results by increasing the average standard deviation. The highest standard deviation in the average percentage gap for the set A instance was found in TS, which was 11.3 on average for each instance. On the other hand, SA+MT produced the best results in terms of solution quality, second only to the GA, requiring significantly less computational effort than GA. The results also showed the metaheuristics converged to a better solution if started from a solution with the MT heuristics rather than starting with a random solution.

This observation becomes even more pronounced for larger Set B and C instances. Figures 8 and 9 show the average and best percentage improvement in maximum trip time of metaheuristics with respect to the MT algorithm for set B and set C instances, respectively, and the MT solution is shown as a dotted line at 0 % on the y-axis. In both these graphs, we can see that, on average, only MT+SA and MT+TS converge to a better solution than the MT heuristic. For SA, TS, and GA, which started from an initial random solution, converged to a worse solution than the MT heuristic, which is apparent as the box for these metaheuristics in both graphs lies below the dotted line.

Furthermore, the graphs highlight the substantial variability in the performance of these metaheuristics for larger instances, as evident from the wide ranges of the boxes and whiskers. For instance, the box plot for SA-500 in Figure 9 shows a wide range, with a maximum improvement of around 176.5% and a minimum of around -87.2% with a mean of -9.4% and a standard deviation of 69.5%. This variability can be attributed to the stochastic nature of these algorithms and their sensitivity to initial solutions (random in this case) and parameter settings.

VI. CONCLUSION

In summary, this paper introduced the MD-RPP-RRV problem, a variant of the arc routing problem, and formulated it as a MILP. It presented SA, TS, and GA metaheuristics designed to solve MR-RPP-RRV by improving the solution quality obtained by the MT heuristic. Optimal solutions were obtained for smaller instances by solving MILP formulation using Gurobi; for larger real-world instances, memory and computing limitations prevented finding optimal solutions. Experimentation revealed that, for smaller instances, SA, TS, and GA significantly improved solution quality, reducing the initial optimality gap of MT heuristic from 162% to 0.9%, 16.2% and 0%, with GA converging to optimal solutions but requiring substantial computational effort. However, for larger instances, SA and TS improved solution quality by 54.1% and 49.9%, respectively, relative to MT, when initialized with MT solutions. Yet, when initialized with a random solution, SA, TS, and GA failed to enhance solution quality for large instances, highlighting the dependence of metaheuristics on the initial heuristic solution for larger instances. Future directions include exploring robust solutions considering multiple vehicle failures, integrating temperature effects on batteries, and addressing facility location and fleet size problems.

ACKNOWLEDGMENT

The authors would like to acknowledge Dr. Menglin Jin from SpringGem for the help and support. They also acknowledge the University of Maryland High-Performance Computing resources (Zaratan) made available for conducting the research reported in this article.

REFERENCES

[1] F. Malin, I. Norros, and S. Innamaa, "Accident risk of road and weather conditions on different road types," *Accident Anal. Prevention*, vol. 122, pp. 181–188, Jan. 2019.

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

[3] E. Sathyamurthy, J. W. Herrmann, and S. Azarm, "Multi-trip algorithm for multi-depot rural postman problem with rechargeable vehicles," 2023, *arXiv:2303.03481*.

[4] Gurobi Optim. LLC. (2023). *Gurobi Optimizer Reference Manual*. [Online]. Available: https://www.gurobi.com

[5] H. A. Eiselt, M. Gendreau, and G. Laporte, "Arc routing problems, Part I: The Chinese postman problem," *Oper. Res.*, vol. 43, no. 2, pp. 231–242, Apr. 1995.

[6] H. A. Eiselt, M. Gendreau, and G. Laporte, "Arc routing problems, Part II: The rural postman problem," *Oper. Res.*, vol. 43, no. 3, pp. 399–414, Jun. 1995.

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

[8] J. K. Lenstra and A. H. G. R. Kan, "On general routing problems," *Networks*, vol. 6, no. 3, pp. 273–280, Jan. 1976.

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

[10] G. Ghiani and G. Laporte, "Location-arc routing problems," *Opsearch*, vol. 38, no. 2, pp. 151–159, Apr. 2001.

[11] T. Liu, Z. Jiang, F. Chen, R. Liu, and S. Liu, "Combined location-arc routing problems: A survey and suggestions for future research," in *Proc. IEEE Int. Conf. Service Oper. Logistics, Informat.*, Oct. 2008, pp. 2336–2341.

[12] L. Levy and L. Bodin, "The arc oriented location routing problem," *INFOR, Inf. Syst. Oper. Res.*, vol. 27, no. 1, pp. 74–94, Jan. 1989.

[13] W. Miehle, "Link-length minimization in networks," *Oper. Res.*, vol. 6, no. 2, pp. 232–243, Apr. 1958.

[14] L. Cooper, "Location-allocation problems," *Oper. Res.*, vol. 11, no. 3, pp. 331–343, Jun. 1963.

[15] L. Cooper, "Heuristic methods for location-allocation problems," *SIAM Rev.*, vol. 6, no. 1, pp. 37–53, Jan. 1964.

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

[17] B. L. Golden, J. S. Dearmon, and E. K. Baker, "Computational experiments with algorithms for a class of routing problems," *Comput. Oper. Res.*, vol. 10, no. 1, pp. 47–59, Jan. 1983.

[18] W. L. Pearn, "Approximate solutions for the capacitated arc routing problem," *Comput. Oper. Res.*, vol. 16, no. 6, pp. 589–600, Jan. 1989.

[19] N. Christofides, "The optimum traversal of a graph," *Omega*, vol. 1, no. 6, pp. 719–732, Dec. 1973.

[20] E. Fernández and J. Rodríguez-Pereira, "Multi-depot rural postman problems," *TOP*, vol. 25, no. 2, pp. 340–372, Jul. 2017.

[21] E. Tirkolaee, A. Hosseinabadi, M. Soltani, A. Sangaiah, and J. Wang, "A hybrid genetic algorithm for multi-trip green capacitated arc routing problem in the scope of urban services," *Sustainability*, vol. 10, no. 5, p. 1366, Apr. 2018.

[22] A. Amberg, W. Domschke, and S. Voß, "Multiple center capacitated arc routing problems: A tabu search algorithm using capacitated trees," *Eur. J. Oper. Res.*, vol. 124, no. 2, pp. 360–376, Jul. 2000.

[23] H. Hu, T. Liu, N. Zhao, Y. Zhou, and D. Min, "A hybrid genetic algorithm with perturbation for the multi-depot capacitated arc routing problem," *J. Appl. Sci.*, vol. 13, no. 16, pp. 3239–3244, Aug. 2013.

[24] A. Kansou and A. Yassine, "New upper bounds for the multi-depot capacitated arc routing problem," *Int. J. Metaheuristics*, vol. 1, no. 1, p. 81, 2010.

[25] W. Yu, Y. Liao, and Y. Yang, "Exact and approximation algorithms for the multi-depot capacitated arc routing problems," *Tsinghua Sci. Technol.*, vol. 28, no. 5, pp. 916–928.

[26] H. Chen, T. Cheng, and J. Shawe-Taylor, "A balanced route design for min-max multiple-depot rural postman problem (MMMDRPP): A police patrolling case," *Int. J. Geographical Inf. Sci.*, vol. 32, no. 1, pp. 169–190, Jan. 2018.

[27] G. N. Frederickson, M. S. Hecht, and C. E. Kim, "Approximation algorithms for some routing problems," in *Proc. 17th Annu. Symp. Found. Comput. Sci. (SFCS)*, Oct. 1976, pp. 216–227.

[28] D. Ahr and G. Reinelt, "New heuristics and lower bounds for the min-max k-Chinese postman problem," in *Proc. 10th Annu. Eur. Symp. Algorithms*, Rome, Italy. Berlin, Germany: Springer, pp. 64–74.

[29] J. Brandão and R. Eglese, "A deterministic tabu search algorithm for the capacitated arc routing problem," *Comput. Oper. Res.*, vol. 35, no. 4, pp. 1112–1126, Apr. 2008.

[30] P. Lacomme, C. Prins, and M. Sevaux, "A genetic algorithm for a bi-objective capacitated arc routing problem," *Comput. Oper. Res.*, vol. 33, no. 12, pp. 3473–3493, Dec. 2006.

[31] M. Dorigo, M. Birattari, and T. Stutzle, "Ant colony optimization," *IEEE Comput. Intell. Mag.*, vol. 1, no. 4, pp. 28–39, Nov. 2006.

[32] E. B. Tirkolaee, M. Alinaghian, A. A. R. Hosseinabadi, M. B. Sasi, and A. K. Sangaiah, "An improved ant colony optimization for the multi-trip capacitated arc routing problem," *Comput. Electr. Eng.*, vol. 77, pp. 457–470, Jul. 2019.

[33] M. Polacek, K. F. Doerner, R. F. Hartl, and V. Maniezzo, "A variable neighborhood search for the capacitated arc routing problem with intermediate facilities," *J. Heuristics*, vol. 14, no. 5, pp. 405–423, Oct. 2008.

[34] A. Amaya, A. Langevin, and M. Trépanier, "The capacitated arc routing problem with refill points," *Oper. Res. Lett.*, vol. 35, no. 1, pp. 45–53, 2007.

[35] A. Hertz and M. Mittaz, "A variable neighborhood descent algorithm for the undirected capacitated arc routing problem," *Transp. Sci.*, vol. 35, no. 4, pp. 425–434, Nov. 2001.

[36] A. Hertz, G. Laporte, and M. Mittaz, "A tabu search heuristic for the capacitated arc routing problem," *Oper. Res.*, vol. 48, no. 1, pp. 129–135, Feb. 2000.

[37] P. Lacomme, C. Prins, and W. Ramdane-Chérif, "A genetic algorithm for the capacitated arc routing problem and its extensions," in *Proc. Workshops Appl. Evol. Comput.* Berlin, Germany: Springer, 2001, pp. 473–483.

[38] J. M. Belenguer and E. Benavent, "A cutting plane algorithm for the capacitated arc routing problem," *Comput. Oper. Res.*, vol. 30, no. 5, pp. 705–728, Apr. 2003.

[39] F. L. Usberti, P. M. França, and A. L. M. França, "Branch-and-bound algorithm for an arc routing problem," in *Proc. Annals XLIV SBPO*, Rio de Janeiro, Brazil, 2012, p. 652.

[40] A. Corberán, I. Plana, and J. M. Sanchis, "A branch and cut algorithm for the windy general routing problem and special cases," *Networks*, vol. 49, no. 4, pp. 245–257, Jul. 2007.

[41] T. Bektaş, G. Erdoğan, and S. Røpke, "Formulations and branch-and-cut algorithms for the generalized vehicle routing problem," *Transp. Sci.*, vol. 45, no. 3, pp. 299–316, Aug. 2011.

[42] A. H. Land and A. G. Doig, "An automatic method for solving discrete programming problems," in *50 Years of Integer Programming 1958–2008*. Berlin, Germany: Springer, 1958, pp. 105–132, doi: 10.1007/978-3-540-68279-0_5.

[43] E. Semsch, M. Jakob, D. Pavlicek, and M. Pechoucek, "Autonomous UAV surveillance in complex urban environments," in *Proc. IEEE/WIC/ACM Int. Joint Conf. Web Intell. Intell. Agent Technol.*, vol. 2, Sep. 2009, pp. 82–85.

[44] A. Puri, "A survey of unmanned aerial vehicles (UAVs) for traffic surveillance," Ph.D. dissertation, Dept. Comput. Sci. Eng., Univ. South Florida, Tampa, FL, USA, 2005, pp. 1–29.

[45] K. S. Lee, M. Ovinis, T. Nagarajan, R. Seulin, and O. Morel, "Autonomous patrol and surveillance system using unmanned aerial vehicles," in *Proc. IEEE 15th Int. Conf. Environ. Electr. Eng. (EEEIC)*, Jun. 2015, pp. 1291–1297.

[46] K. Sundar and S. Rathinam, "Algorithms for routing an unmanned aerial vehicle in the presence of refueling depots," *IEEE Trans. Autom. Sci. Eng.*, vol. 11, no. 1, pp. 287–294, Jan. 2014.

[47] K. Yu, A. K. Budhiraja, and P. Tokekar, "Algorithms for routing of unmanned aerial vehicles with mobile recharging stations," in *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, May 2018, pp. 5720–5725.

[48] Y. Choi, B. Robertson, Y. Choi, and D. Mavris, "A multi-trip vehicle routing problem for small unmanned aircraft systems-based urban delivery," *J. Aircr.*, vol. 56, no. 6, pp. 2309–2323, Nov. 2019.

[49] K. Dorling, J. Heinrichs, G. G. Messier, and S. Magierowski, "Vehicle routing problems for drone delivery," *IEEE Trans. Syst., Man, Cybern., Syst.*, vol. 47, no. 1, pp. 70–85, Jan. 2017.

[50] Y. Choi, Y. Choi, S. I. Briceno, and D. N. Mavris, "An extended savings algorithm for UAS-based delivery systems," in *Proc. AIAA SciTech Forum*, 2019, p. 1796.

[51] E. W. Dijkstra, "A note on two problems in connexion with graphs," *Numerische Math.*, vol. 1, no. 1, pp. 269–271, Dec. 1959.

[52] S. Kirkpatrick, C. D. Gelatt Jr., and M. P. Vecchi, "Optimization by simulated annealing," *Science*, vol. 220, no. 4598, pp. 671–680, 1983.

[53] C. W. Ahn and R. S. Ramakrishna, "Elitism-based compact genetic algorithms," *IEEE Trans. Evol. Comput.*, vol. 7, no. 4, pp. 367–385, Aug. 2003.

[54] DJI. (2022). *Mavic 2—Product Information*. [Online]. Available: https://www.dji.com/mavic-2/info

[55] E. Sathyamurthy, "Multi-flight algorithms for multi-UAV arc routing problem," M.S. thesis, Dept. Mech. Eng., Univ. Maryland, College Park, MD, USA, 2021.

EASHWAR SATHYAMURTHY received the B.Tech. degree from Jawaharlal Nehru Technological University (JNTU) Hyderabad, Telangana, India, in 2019, and the M.Eng. and M.S. degrees in systems engineering from the University of Maryland (UMD), College Park, MD, USA, in 2020 and 2021, respectively, where he is currently pursuing the Ph.D. degree in mechanical engineering. His research interests include path planning of multi-agent autonomous systems, engineering optimization, and decision-making under uncertainty.

JEFFREY W. HERRMANN received the B.S. degree in applied mathematics from Georgia Institute of Technology, Atlanta, GA, USA, in 1990, and the Ph.D. degree in industrial and systems engineering from the University of Florida, Gainesville, FL, USA, in 1993.

He is currently the St. Abbo of Fleury Endowed Chair in Engineering and a Professor with the Department of Mechanical Engineering, The Catholic University of America. He has published over 100 journal articles and refereed conference papers and 15 book chapters, coauthored an engineering design textbook, edited two handbooks, authored a textbook on engineering decision-making and risk management, and written a book on metareasoning. His current research interest includes metareasoning for autonomous systems.

Dr. Herrmann is a member of the Society of Catholic Scientists, IISE, ASME, and the Design Society. He is an Associate Editor of the *Journal of Autonomous Vehicles and Systems*.

SHAPOUR AZARM received the Ph.D. degree from the University of Michigan, Ann Arbor, MI, USA. He is currently a Professor with the Department of Mechanical Engineering and an Affiliate Professor with the Applied Mathematics and Statistics, and Scientific Computation Program, University of Maryland, College Park, MD, USA. He is a Senior Advisor of *Structural and Multidisciplinary Optimization*. His research interests include predictive modeling, engineering optimization, and decision analysis. He is a fellow and a Life Member of the American Society of Mechanical Engineers. He was the Editor-in-Chief of the *Journal of Mechanical Design*.