A Learning Based Algorithm for Drone Routing

Umut Ermağan[a], Barış Yıldız[a], F. Sibel Salman[a,*]

[a] *College of Engineering, Koç University, Sariyer, Istanbul 34450, Turkey*

Abstract

We introduce a learning-based algorithm to solve the drone routing problem with recharging stops that arises in many applications such as precision agriculture, search and rescue, and military surveillance. The heuristic algorithm, namely Learn and Fly (L&F), learns from the features of high-quality solutions to optimize recharging visits, starting from a given Hamiltonian tour that ignores the recharging needs of the drone. We propose a novel integer program to formulate the problem and devise a column generation approach to obtain provably high-quality solutions that are used to train the learning algorithm. Results of our numerical experiments with two groups of instances show that the classification algorithms can effectively identify the features that determine the timing and location of the recharging visits, and L&F generates energy feasible routes in a few seconds with around 5% optimality gap on the average.

Keywords: Drone routing, learning based algorithm, column generation, machine learning

1. Introduction

The use of unmanned aerial vehicles (UAVs), commonly known as drones, is on the rise in many fields such as small-parcel delivery, humanitarian logistics, surveillance, and environmental monitoring, in addition to military applications. Precision agriculture is one of the application areas that inspires us to study the optimal routing of drones. Precision agriculture can be described briefly as the usage of information technologies for agricultural practices to improve profitability and sustainability in production (Gebbers and Adamchuk, 2010). Precision agricultural systems provide scientific and data-driven support for crop management. As environmental conditions, such as rainfall, humidity, and temperature, can easily affect agricultural products, the farmers need to get reliable information on soil and crops to take precautionary actions (Swain et al., 2007). For that purpose, drones can be effectively used to capture aerial images to be processed digitally to extract valuable information about the crops' status, such as vegetation density, crop stress, and water deficit, or measure various key parameters via remote sensing. Compared to planes and satellites, drones have several advantages in aerial imaging, such as lower cost, higher availability, low altitude (below the cloud cover), and low-speed flight capabilities, which promote their use in precision agriculture, as well as in many other

*Corresponding author
Email address: ssalman@ku.edu.tr (F. Sibel Salman)

application areas (Barrientos et al., 2011). In systems that utilize images captured by drones, the drone is equipped with a camera, a positioning system, storage memory, a wireless transceiver, and a battery for the system to work autonomously. In order to minimize human intervention, a fast-running routing algorithm is needed to guide the drone over the fields. An optimal altitude is selected for the drone in the allowed flight range, considering the trade-off between image resolution and battery consumption. The drone's route consists of flight legs, each having one or more image capturing (surveillance) tasks between recharging stops at recharging stations (RSs) as the battery is depleted frequently. The route itself includes all the surveillance tasks and some visits to RSs along the way.

This study aims to develop an efficient algorithm to determine the route of the drone while considering the limited battery capacity that mandates conducting multiple route legs in between recharge station stops. In particular, the algorithm should have a sufficiently small running time to allow rerouting when conditions (e.g., wind, cloud shadows, new information gained during the surveillance) change dynamically. A network is constructed as an input to the problem by defining nodes for both surveillance tasks and the RS locations. Here we want to underline that surveillance tasks may require visiting multiple points in a region, although a single node represents them in the network. Therefore, distances between the nodes include the time it takes for the drone to move from one region to the other and the time it takes to complete the surveillance tasks in a region (i.e., visiting a sequence of points in it). The drone should visit one of the RSs before it runs out of battery and then takes off again to continue its route until all required tasks are completed in minimum time. We develop a learning-based fast algorithm: Learn and Fly (L&F), which can solve the Drone Routing Problem (DRP) in real-time, in a setting that allows the routing decisions to be updated each time operational conditions change. A standard arc-based integer programming (IP) formulation of the routing problem falls short of obtaining high-quality solutions to implement or strong bounds to measure the quality of a given feasible solution. Therefore, we present a novel path-segment and cycle based formulation to model the routing problem, motivated by the fact that such formulations have been proposed in various other forms and successfully implemented in recent years for other problems (Yıldız and Karaşan, 2017; Yıldız and Savelsbergh, 2019). Using this formulation, we develop a column generation algorithm to obtain high quality solutions (i.e., having less than %2.1 optimality gap on the average for the problem instances we study) and strong lower bounds on the total route time. Furthermore, we use these solutions to train the learning algorithm so that it can solve the online routing problem in a few seconds. We test the performance of our algorithm on two groups of instances having varying sizes. We observe that the algorithm gives near-optimal solutions in a few seconds on data sets with up to 40 nodes (i.e., 40 tasks). Even if the training set is based on smaller sized instances, L&F still achieves high performance.

In the next section we provide a brief literature review on static and dynamic routing problems related to UAVs/drones. In Section 3 we present the problem description, the integer programming model, and the column generation procedure. We propose our learning based heuristic algorithm in Section 4. In Section 5, instance generation, computational results and performance analysis of the algorithm are presented. Finally, concluding remarks are given in Section 6.

2. Literature Review

Our problem falls into the broad class of Vehicle Routing Problems (VRP). It focuses on finding a single-vehicle route, such as the Travelling Salesman Problem (TSP), but it differs from it as follows. Due to the battery consumption and its limited capacity, we cannot visit all the nodes in a single tour. Therefore, the route should consist of sequential flight legs in between RSs such that the drone should start its next leg from the RS it landed in the previous leg. Visiting one or more RSs more than once, the route may contain cycles. Moreover, our capacity constraint is in terms of the battery level and differs from the standard capacitated VRP models in the following way. The remaining battery should be updated after completing each task and stopping at an RS since the remaining capacity decreases and increases along the route correspondingly. The routing of a single drone or multiple drones has been studied as variants of the TSP and VRP. For the growing literature on the topic, we refer the interested reader to comprehensive reviews by Otto et al. (2018) about the recent advances in optimization algorithms and Khoufi et al. (2019) about using TSP and VRP approaches with new transportation technologies and business models. Here, we focus on recent studies involving dynamic settings similar to the one we consider or involving methodologies similar to those we propose.

The use of drones for surveillance tasks, with obvious advantages, attracts significant attention from both the industry and academia. However, it has been pointed out that there are not enough studies in the literature to deal with the novel algorithmic challenges to realize this potential, mostly due to the complexity of the underlying dynamic routing problem, which limits the use of existing methods to guide online decision making (Crainic et al., 2009). Solution approaches for combinatorial optimization problems based on learning have recently emerged. Modaresi et al. (2020) benefit from learning to solve combinatorial optimization problems with instances that are unknown in advance. They study a combinatorial optimization problem with a single period. Their main idea in the learning is based on the feedback obtained from the previous solutions, while machine learning algorithms carry out the learning in our study. In our study, the learning phase is performed as pre-processing, and its outputs are utilized to guide key decisions in the proposed algorithm. Cui et al. (2018) address a path planning problem in which the future actions of a UAV can be decided. In the study, a Markov Decision Process (MDP) model is used to handle some dynamic uncertainties such as control strategies, flight environments, and any other bursting-out threats because of the unstructured environment. They propose a path-planning model based on Q-learning that obtains high tracking accuracy in the solutions. Q-learning is a reinforcement learning by which the UAV can learn an optimal strategy by experiencing the consequences of the previous actions. Since Q-learning aims to maximize the expected reward at each position of the UAV with the help of the previous actions, it needs more samples to learn. On the other hand, in our study, we use supervised learning algorithms instead of reinforcement learning algorithms like Q-learning. The most important reason is that we can address the drone routing problem as a classification problem in which the output variables, i.e., the target labels, can be categorized. The supervised machine learning algorithms identify a function mapping the input variables, i.e., the features, to the target labels. In our problem, after obtaining the function

mapping the features, these algorithms can produce correct labels for the new samples with high accuracy. Hence, we avoid the large data and computation requirements for the Q-learning. We refer the interested readers to the studies by Watkins and Dayan (1992) and Sutton and Barto (2018) for detailed descriptions of Q-learning and reinforcement learning.

Several UAV routing studies consider uncertain parameters. Chow (2016) studies a UAV routing problem in which the UAVs monitor traffic under uncertainty using a stochastic policy. The study, which aims to minimize the routing cost of a certain number of UAVs, provides a new policy by re-formulating the Bellman equation, as well as an approximate algorithm. The results obtained from the developed algorithm are compared against static and myopic policies. Kim et al. (2018) take the drone routing problem into account under uncertainty in the battery capacity subject to air temperature changes. A robust optimization approach utilizing uncertainty sets is proposed. Al-Sabban et al. (2013) identify Markov Decision Processes to find an optimal energy-based path of a single UAV while minimizing total travel time and power consumption. Hybrid Gaussian distribution is used to find the direction and magnitude of the wind. Hence, UAV's next action can be decided by considering the wind effect. Similarly, Thibbotuwawa et al. (2020) consider the weather conditions during planning the flight missions of UAVs. The goal of the study is to meet the demand of each customer within a time horizon. They propose a declarative model for the UAVs' flight mission planning. The solution approach is based on finding the flight mission approximately at two levels. One of them is the mission planning level, and the other is the sub-mission planning level. In the sub-mission planning level, the mission area is subdivided according to the UAVs' relative capacities. The solution is obtained via the proposed model for each sub-divided area. The combined solution of the sub-mission solutions represents the mission plan of the UAVs.

In addition to methods based on probabilistic policies (e.g., Chow (2016), Kim et al. (2018), Al-Sabban et al. (2013)), some deterministic algorithms have also been studied. Guerriero et al. (2014) consider multiple UAVs covering events that occur over time in a finite area. Each event randomly occurs at a location during a time window, such as the events during a sports action. They develop a multi-criteria optimization model, in which the criteria are to minimize the total distances traveled by the UAVs, maximize the customer satisfaction by arriving on time to capture each event and minimize the number of used UAVs. They pose the problem as an instance of the VRP with Soft Time Windows (VRP-STW) and test both an epsilon-constraint approach to determine an approximation of the Pareto front and a rolling horizon approach such that when the problem is solved at a time point, only the known events are considered and no information is given on future events. Li et al. (2018) propose a heuristic algorithm to solve the scheduling problem of UAVs for traffic monitoring considering the uncertain monitoring demands. They formulate a mixed integer linear programming model by combining the capacitated arc routing problem with the inventory-routing problem to be able to monitor the traffic with a time-dependent inventory-based method. By a local branching based method, the best objective value is chosen among the solution spaces found with the branching cuts when the maximum iteration number is reached or there is no improvement.

Electric vehicles have similar features with drones in terms of the battery limit and the need for

recharging (see Erdelić and Carić (2019) for a comprehensive review of the burgeoning literature on electric vehicle routing). While electric vehicles are subject to a road infrastructure, this is not the case for drones. Direct flights take place between the points to be covered unless restricted areas are on the way. We propose a method to avoid the restricted areas in an efficient way while generating the distance matrix. The wind factor can also be incorporated by adjusting the input travel times according to the wind direction and speed. Although the electric vehicle routing problems tolerate moderate run times, drone routing requires particularly fast solution methods due to the need to rerun the solution algorithm in real-time when the current conditions, such as the wind effect, change.

In the literature, different approaches have been developed to solve the electric vehicle routing problem (EVRP). Early studies (e.g., Schneider et al. (2014), Goeke and Schneider (2015), Hiermann et al. (2019)) utilize metaheuristic algorithms. Sassi and Oulamara (2017) develop an exact method based on two phases, in addition to a sequential heuristic and a global heuristic to solve larger instances. On the other hand, different exact solution methodologies have also been utilized (e.g. Desaulniers et al. (2016), Liao et al. (2016), Yıldız et al. (2016)). However, to the best of our knowledge, learning methods have not been used for the EVRPs.

2.1. Our contributions

With respect to the existing studies, the contribution of this paper is threefold.

- We propose a new integer programming (IP) formulation for DRP, coupled with a column generation approach. The linear programming (LP) relaxation of the formulation yields strong lower bounds, as evidenced in our computational study. Moreover, the integer solutions obtained by considering only the columns generated at the time the LP relaxation is solved provide high-quality solutions to train our learning algorithm.

- We propose an innovative approach that "learns" from the features of high-quality solutions obtained via the column generation algorithm and guides the heuristic on recharging decisions. Building on a TSP solution that disregards the drone's recharging needs, the solution is rendered energy feasible according to the learned recharging actions. Thereby, near-optimal solutions can be obtained in a fraction of a second, starting from the TSP solution. To the best of our knowledge, this study is the first one to develop a learning-based approach, utilizing a column generation procedure, to solve the DRP.

- We conduct numerical experiments to test the suggested method's effectiveness and gain important insights about algorithm design to solve the DRP and its extensions. In particular, our results indicate that:

  - The learning-based algorithm L&F can generate solutions with around 5% optimality gap on the average, for the instances we consider in our experiments with up to 40 nodes.
  - Trained on the solutions of small-sized problem instances, L&F can still successfully find high quality solutions for the large-sized problem instances.

– We are able to identify which features contribute to the learning accuracy the most and hence these findings provide hints to guide the development of new approaches for drone routing and its extensions.

3. Problem Definition, Model Formulation and Column Generation

This section provides the formal definition of the DRP, introduces our IP formulation to model it, and presents the column generation (CG) procedure we use to obtain high-quality solutions (to train the learning algorithm) and lower bounds on the optimal solutions.

3.1. Notation and Problem Definition

Let N be the set of nodes to be included into the route of the drone and R be the set of recharging stations, where the drone can fully recharge its battery or swap it with a fully charged one. DRP is defined on a complete directed graph $G = (V, A)$, with node set $V = (N \cup R)$ and arc set $A = \{(i, j) : i \in V, j \in V \setminus \{i\}\}$. An arc $(i, j) \in A$ is called a flight arc if $j \in N$ and it is called a recharging arc, otherwise. The travel time associated with a flight arc $(i, j) \in A$ is denoted by t_{ij} and it includes the time spent for repositioning the drone and the time it takes for it to complete task $j \in N$. On the other hand, for a recharging arc (i, j) the duration t_{ij} includes only the time it takes for the drone to reach the recharging station j. The time it takes to recharge/swap batteries is denoted by t_r. The range of the drone, i.e., the allowed flight time with a fully charged battery, is denoted by τ. We call a route feasible, if the flight time between any recharging stops in it is less than the range of the drone, τ.

The DRP aims to find the minimum length feasible route in G, that starts from an arbitrary recharging station $o_w \in R$, visits all the nodes in N and ends in an arbitrary recharging station $d_w \in R$.

3.2. Path-segment and Cycle Formulation

The idea to build routes by concatenating path-segments, each representing the route segment between replenishment stops, has been successfully used to model and solve routing problems with replenishment that arise in many transportation and telecommunications applications (Yıldız and Karaşan, 2017; Yıldız et al., 2016; Yıldız and Savelsbergh, 2019). Extending this idea, here we consider building the route of the drone by concatenating simple paths and cycles, where the drone stops at a RS at the end of each path/cycle to replenish its batteries to start a new path/cycle.

Our formulation is built with path-segments which are directed simple paths in G, where the start and destination nodes o_p and d_p of path p are recharging stations. A path-segment p is called feasible if its length $t_p = \sum_{(i,j)\in p} t_{ij}$ is less than the range of the drone (τ). The set of nodes in path-segment p is denoted by $N(p)$ and the set of all feasible path-segments in G is denoted by P. For notational convenience, we denote the set of all feasible path-segments that visit a node $i \in N$ with $P(i)$. For a set $S \subset R$, we define $\delta^-(S) = \{p \in P : o_p \in S, d_p \in R \setminus S\}$.

In addition to the simple paths, we also consider simple cycles in our formulation where only one node, which is called the origin, is visited more than once. In our formulation, we consider only simple cycles where the origin is a recharging station. A simple cycle c is called feasible if its length $t_c = \sum_{(i,j)\in c} t_{ij}$ is at most the drone's range, τ. The origin of a cycle c is denoted by o_c and the set of all nodes visited by the cycle is denoted by $N(c)$. The set of all feasible simple cycles in G are denoted by C and the set of all feasible simple cycles that visit a node $i \in N$ is shown by $C(i)$.

Table 1: Notation used in the formulation

Sets	**Description**
V	Set of nodes consisting of nodes that are required to be visited and RSs
N	Set of nodes that are required to be visited
R	Set of RSs
S	Subsets of RSs
A	Set of arcs
P	Set of simple paths
C	Set of simple cycles
$P(i)$	Set of simple paths that include node i
$C(i)$	Set of simple cycles that include node i
$\delta^-(S)$	Set of simple paths whose starting RSs (o_p) are in the subset S, but ending RSs (d_p) are not in the subset S
Parameters	**Description**
t_{ij}	Travel time associated with arc (i,j)
d_{ij}	Euclidean distance associated with arc (i,j)
t_p	Travel time of simple path p
t_c	Travel time of simple cycle c
t_r	Recharge time of the drone at recharge station $r \in R$
τ	Total flight time of the drone with fully charged battery
Decision Variables	**Description**
x_p	Binary variable that indicates whether a path-segment $p \in P$ is used
y_c	Binary variable that indicates whether a simple cycle $c \in C$ is used
z_r	Binary variable that indicates whether a recharging station $r \in R$ is visited
s_r	Binary variable that indicates whether the drone starts its tour at $r \in R$
e_r	Binary variable that indicates whether the drone ends its tour at $r \in R$

With the sets, parameters and decision variables defined in Table 1, we develop our path-segment and cycle formulation $\mathcal{P}_{\mathcal{DRP}}$ as follows.

$$\min \sum_{p\in P}(t_p + t_r)x_p + \sum_{c\in C}(t_c + t_r)y_c \tag{1}$$

s.t.

$$\sum_{r\in R} s_r = 1 \tag{2}$$

$$\sum_{r\in R} e_r = 1 \tag{3}$$

$$\sum_{p \in P(i)} x_p + \sum_{c \in C(i)} y_c = 1 \qquad \forall i \in N \tag{4}$$

$$e_r - s_r + \sum_{p \in P: o_p = r} x_p - \sum_{p \in P: d_p = r} x_p = 0 \qquad \forall r \in R \tag{5}$$

$$x_p \leq z_{o_p} \qquad \forall p \in P \tag{6}$$

$$y_c \leq z_{o_c} \qquad \forall c \in C \tag{7}$$

$$\sum_{p \in \delta^-(S)} x_p \geq z_r - \sum_{\bar{r} \in S} e_{\bar{r}} \qquad \forall S \subset R, r \in S \tag{8}$$

$$x_p \in \{0, 1\} \qquad \forall p \in P \tag{9}$$

$$y_c \in \{0, 1\} \qquad \forall c \in C \tag{10}$$

$$z_r \in \{0, 1\} \qquad \forall r \in R \tag{11}$$

$$s_r \in \{0, 1\} \qquad \forall r \in R \tag{12}$$

$$e_r \in \{0, 1\} \qquad \forall r \in R \tag{13}$$

The objective is to minimize the total flight and recharging time of the drone. Here we note that the objective includes the recharging time at the final stop. Since the solution would not change if one subtracts the recharging time at the last stop from the objective function, we opted to present it in this form for the sake of simplicity. Constraints (2) and (3) determine the start and end nodes of the drone's tour. Since the directed graph $G = (V, A)$ is a complete graph, constraints (4) ensure that each node is visited by the drone exactly once. Constraints (5) are flow balance constraints ensuring that the number of incoming path segments to a recharging station is the same with the number of outgoing path segments (i.e., the drone does not suddenly appear/disappear at a recharging station), except for the recharging stations where the drone starts/ends its tour. Constraints (6) and (7) ensure that a path/cycle can only be used if the origin is an activated recharging point. Constraints (8) eliminate the sub-tours formed by path-segments. To achieve this, for any subset S of recharging stations that contains at least one activated recharging station and does not include the recharging station in which the drone completes its tour (i.e., $\sum_{\bar{r} \in S} e_{\bar{r}} = 0$), they ensure that there is at least one path-segment going out of the subset S (i.e., $\sum_{p \in \delta^-(S)} x_p \geq 1$). Finally, (9)-(13) are the variable domain restrictions.

Note that formulation $\mathcal{P}_{DRP}$ determines the recharge stations that will be utilized, along with the route of the drone. However, due to several practical reasons, such as the cost of establishing a recharging post and to avoid occupation of a part of the fertile land, it is not likely to have a large number of RSs in real world DRP instances. In fact, what necessitates a special treatment of DRP is the scarcity of those recharging stations. Therefore, in the optimal solutions for those realistic problem instances with a small yet strategically located recharging points, it is quite likely that the drone would visit all of the established recharging stations, as we also observe in our numerical experiments. With this observation, one can simplify $\mathcal{P}_{DRP}$ by focusing on those solutions where $z_r = 1$ for all $r \in R$. Note

that with this modification constraints (6) and (7) become redundant, and constraints (8) simplify to:

$$\sum_{p \in \delta^-(S)} x_p + \sum_{\bar{r} \in S} e_{\bar{r}} \geq 1 \qquad \forall S \subset R, \tag{14}$$

Obviously, a similar simplification would be possible if some of the variables z are set to zero, meaning that those stations will not be visited by the drone. Considering the significant computational advantages of such a simplification, solving $\mathcal{P}_{DRP}$ for each active recharge station combination and then choosing the one with the best result becomes a computationally viable option when the number of recharging stations is small, as one would expect in realistic problem instances. Although the column generation approach we describe next can be used for the original formulation (1)-(13), for the sake of simplicity, from here on, when we mention $\mathcal{P}_{DRP}$ we refer to the simplified formulation: (1)-(5), (9), (10) and (12)-(14).

Clearly, the number of feasible path-segments $|P|$ and the number of feasible simple cycles $|C|$ grow exponentially with the number of nodes to be visited by the drone and it is not possible to solve $\mathcal{P}_{DRP}$ directly for realistic size problem instances. However, contrary to compact "arc-based" formulation alternatives in which big M type constraints are needed to keep track of the energy level of the drone and eliminate subtours, $\mathcal{P}_{DRP}$ can provide strong linear relaxation bounds and solving it with a "good" subset of path and cycle variables can warrant high quality solutions with small optimality gaps (Yıldız and Savelsbergh, 2019). Both of these outputs are critical to develop (train and test) the learning based solution method we propose in this study and can be achieved by solving the linear relaxation LP_{DRP} with a column generation approach that we present in the next section.

3.3. Column Generation Algorithm

We start the column generation algorithm (CG) with a restricted master formulation that contains a subset of path and cycle variables and extend it iteratively by adding new energy feasible path-segments and simple cycles as needed. More precisely, at the k^{th} iteration of the CG, we solve the restricted formulation LP^k_{DRP}, which contains the path and cycle variables added in the previous iterations, and then look for path and cycle variables that are not in the restricted master problem but have favorable reduced costs to include them in the model. To identify such variables, we solve two *pricing problems* (one for path variables and one for cycle variables) and terminate the CG when solving the pricing problems does not return path and cycle variables with negative reduced costs. Here we want to note that the number of subtour elimination constraints (8) is a function of the number of recharging stations (not the number of nodes to visit), and thus for the problem instances we consider in this study (which have no more than five recharging stations) it is possible to include all of those constraints in the model a priori. However, from a technical perspective one can also consider a cutting plane approach to include them in the model in an iterative way by using the efficient separation approaches that exist in the literature (Lee et al., 1996).

Pricing problem for the path variables:. Let α, β and λ be the dual variables associated with the constraints (4), (5) and (14), respectively. For a path-segment $p \in P$, the reduced cost π_p of the

variable x_p can be calculated as follows.

$$\pi_p = t_r + \sum_{(i,j)\in p} t_{ij} - \sum_{i\in p} \alpha_i + \beta_{o_p} - \beta_{d_p} + \sum_{S\subset R: d_p\in S, o_p\notin S} \lambda_S \tag{15}$$

Let $P(o,d)$ be the set of feasible path-segments in G that start from the recharging station o and end at the recharging station d, i.e., $P(o,d) = \{p \in P : o_p = o, d_p = d\}$. Note that, for all $p \in P(o,d)$, we can write the reduced cost as $\pi_p = F_{od} + \sum_{(i,j)\in p} \bar{t}_{ij} + t_r$, where $F_{od} = \beta_o - \beta_d + \sum_{S\subset R: d\in S, o\notin S} \lambda_S$ and $\bar{t}_{ij} = t_{ij} - \alpha_i$, for all $(i,j) \in A$. It is easy to see that for an origin-destination pair (o,d), the path-segment with the smallest reduced cost can be found by solving a resource constrained shortest path problem (RCSP; Beasley and Christofides, 1989) on the graph G considering arc costs $\bar{t}_{ij}$ and resource usage t_{ij} for all $(i,j) \in A$. Let $p^*_{od} \in P(o,d)$ be the simple-path with the shortest length $\bar{t}_{p^*} = \sum_{(i,j)\in p^*} \bar{t}_{ij}$. Clearly, if $F_{od} + \bar{t}_{p^*} + t_r < 0$, we can conclude that the variable x_{p^*} needs to be added to the restricted formulation in the next iteration of the CG, and $F_{od} + \bar{t}_{p^*} + t_r \geq 0$ implies that $\pi_p \geq 0$ for all $p \in P(o,d)$. The mixed integer programming (MIP) model we use to solve RCSP is presented in Appendix A. We note that solving RCSP by MIP suffices for our purposes as the run times are reasonable for the size of the problems that we solve. For larger instances, specialized RCSP algorithms may be utilized instead of solving the MIP.

Pricing problem for the cycle variables:. The reduced cost π_c of a cycle variable y_c, $c \in C$ can be calculated as

$$\pi_c = t_r + \sum_{(i,j)\in c} t_{ij} - \sum_{i\in c} \alpha_i, \tag{16}$$

where α is the unrestricted dual variable associated with Constraints (4), as defined before.

Note that using a similar cost transformation for the arcs we used in the previous case, i.e., setting $\hat{t}_{ij} = t_{ij} - \alpha_i$, for all $(i,j) \in A$, we can solve the pricing problem to detect cycle variables with negative reduced costs by solving RCSP problems on the *pricing* graphs $G_o = (\hat{V}, \hat{A})$, $o \in R$, which we define as follows.

For each pricing graph $G_o = (\hat{V}, \hat{A})$, we create a copy of the recharging station o, denoted as $\hat{o}$, and define $\hat{V} = V \cup \{\hat{o}\}$. Similarly, we define the arc set as $\hat{A} = A \cup \{(\hat{o}, i) : i \in V \setminus \{o\}\} \cup \{(i, \hat{o}) : i \in V \setminus \{o\}\}$. For the "original" arcs $(i,j) \in A$, $\hat{t}_{ij}$ is the cost and t_{ij} is the resource usage. For the "artificial" arcs, the costs and the resource usages are the same with the associated original arcs (i.e., $\hat{t}_{\hat{o},i} = \hat{t}_{o,i}$, $\hat{t}_{i,\hat{o}} = \hat{t}_{i,o}$, $t_{\hat{o}i} = t_{oi}$ and $t_{i\hat{o}} = t_{io}$).

Clearly, if the solution of the RCSP problem from o to $\hat{o}$ in G_o, with arc costs $\bar{t}$ and resource usage t has an optimal solution value that is not smaller than $-t_r$, we can conclude that there is no cycle that starts from o, for which the variable y_c has a negative reduced cost. Otherwise, the solution of the RCSP problem returns a cycle c^* such that y_{c^*} has a negative reduced cost and needs to be added to the restricted formulation in the next iteration of the CG.

As explained above, we solve the pricing problem at each iteration k of the CG by solving $|R| \times (|R|-1) + |R|$ RCSP problems (one for each origin-destination pair $(o,d) \in R \times R$ for the path-segment

variables, and for each recharging station $o \in R$ for the cycle variables). If any of those RCSP solutions produce a path or cycle variable with negative reduced costs, we add the associated variables to the restricted model LP^k_{DRP} and move to the next iteration. Otherwise, we terminate the CG declaring the solution of LP^k_{DRP} as the optimal solution of LP_{DRP}.

Once LP_{DRP} is solved, we then apply the following heuristic (Behnke et al., 2020) to find high quality solutions for the $\mathcal{P}_{DRP}$ by solving it by considering only those path and cycle variables generated throughout the CG iterations. As we discuss in more detail when we present the results of our numerical experiments, such a procedure can provide high quality solutions (with 2% optimality gaps on the average) that can be used to train our learning algorithm as we explain next.

4. Learn and Fly Algorithm

The main idea behind the learning based algorithm L&F is to first construct a *giant tour* that visits all the nodes without considering the recharging needs of the drone and then insert recharging stops by learning how to do so from the high quality solutions. With the help of the learning outputs, at any point during the giant tour, it is decided whether to visit an RS or not before moving to the next node. More precisely, we use the learning approach to decide when the drone will visit which RS to recharge its battery. Because the prediction takes very short time, L&F essentially reduces DRP to solving merely a TSP for which there exists several approaches that can produce good solutions in very short run times (Dorigo et al. (1996), Focacci et al. (2002)). As we discuss in detail in Section 5, where we present the results of our numerical experiments, such a two step approach, namely solving the TSP without considering the recharging needs and then inserting the recharging stops on the resulting tour can indeed provide good solutions (i.e., solutions with around 5% optimality gaps for instances with 40 nodes). Since L&F can solve the DRP in very short time (e.g., around 6s for instances with 40 nodes), we consider multiple giant tours to choose the best solution among them.

As an alternative for L&F, we present a mathematical formulation (see Appendix B) to decide when to deviate from the giant tour to visit a recharging station. When we solve the DRP with this mathematical model, with a commercial solver such as CPLEX, we observe that it takes around 431 seconds on average for instances with 40 nodes (surveillance tasks). The difference between the solutions found by the model and the solutions found by L&F is quite small (i.e., 1.55% on average). For larger instances, updating the solutions with the mathematical model as the environmental conditions change is not possible while the drone hovers over a point because of its battery limit. For example, obtaining a solution for instances with 60 nodes takes around 77 minutes, while L&F finds the solution in around 7 seconds.

Next, we discuss the features we consider to learn the timing and locations of recharging stops and their relative importance to make accurate predictions, as well as the alternative supervised machine learning algorithms that can be used for the task.

4.1. Feature Selection

Clearly, the most important step of devising a learning algorithm is determining the feature set (variables) to base the predictions. During the feature selection process, one decides which features are to be measured in light of learning targets. In our case, the target labels are visiting or not visiting an RS and our features should be closely related to the circumstances that affect the replenishment time and place. Along this goal, we selected a diverse set of features that can be divided into three groups. The first group consists of the features that affect the current battery capacity of the drone, i.e. features F1, F2, F3, and F4 in Table 2. The second group is related to the factors that affect the battery capacity in the next step determined by the decision of whether to visit an RS or not. Features F5, F6, F7, and F8 constitute this group. The third group with only one member, feature F9, helps L&F to make recharging decisions by considering how many nodes are left to complete the tour. As we discuss more in detail in Section 5 when we present the results of our numerical experiments, all these features collectively play important roles to provide high accuracy predictions, even though the battery level (F1) and extra distance (F4) have the highest impact on the predictions.

Table 2: Definitions of the Features

ID	**Feature**	**Definition**
1	Battery Level	The current battery level
2	Current-Next	The distance between the current node and the next node in the giant tour
3	Current-Nearest	The distance between the current node and the nearest node to it
4	Extra Distance	The extra distance covered by the drone if it visited the recharge station instead of visiting the next node in the giant tour
5	Next-RS	The distance between the next node in the giant tour and the nearest recharge station to it
6	Nearest-RS	The distance between the nearest node to the current node and the nearest recharge station to it
7	Next-Second	The distance between the two successive nodes after the current node in the giant tour
8	Second-RS	The distance between the second node after the current node in the giant tour and the nearest recharge station to it
9	Unvisited Nodes	The number of nodes that the drone has not visited yet

4.2. The Choice of the Supervised Machine Learning Algorithm

One of the critical steps is to determine which supervised machine algorithm is best able to characterize the training set. The algorithms that are judged to be satisfactory with the preliminary tests can be used as classifiers (Kotsiantis et al., 2007). Choosing a classifier from the learning algorithms and predicting a classifier's accuracy rate in the real world data sets are strongly related to the final accuracy of a classifier. Therefore, estimating a consistent classification accuracy is significant for the

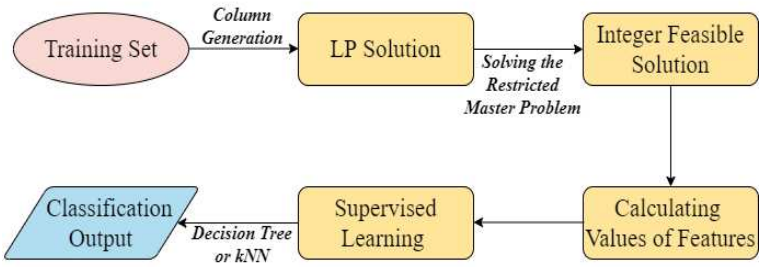
success of learning (Kohavi (1995)). As in most studies on learning, we also prefer to evaluate the learning model via the k-fold cross validation method. In this method, the data set is split into k subsets such that each of them has almost the same size. While k-1 subsets constitute the training set, the remaining one is used as the validation set, and this process is repeated k times for each subset to reduce bias and variance.

After testing several supervised machine algorithms (e.g., Support Vector Machine, Linear Regression, Naive Bayes and Neural Networks), we preferred to use the Decision Tree and k-Nearest Neighbors (kNN) algorithms as two different approaches for the learning phase in the L&F, as they resulted in the highest classification accuracy in almost every tested instance of the DRP.

4.3. Learn and Fly Procedure

For the learning phase, we utilize the feasible integer solutions obtained from the CG as the training set to understand when the drone is replenished. The training set in our study consists of solutions to 500 instances randomly generated for each instance size as explained in Subsection 5.1. We note that the execution time of the learning phase does not affect the execution time of the L&F. After obtaining the IP solutions for the training set, the values of the features listed in Table 2 are calculated. Then, as shown in Figure 1, the two selected supervised machine learning algorithms, as two alternatives, are used to conduct the learning phase.

Figure 1: Steps of Learning



A decision tree is a tree-like structure constructed with data partitioning in a recursive manner called "recursive partitioning". In each iteration, each child node, i.e. each leaf, represents the outcome of the test while the branches represent the decision rules. In the decision tree learning, each internal node is split into two child nodes via the chosen split criterion named as "Attribute (Feature) Selection Measures (ASM)" (Aruna, 2013; Nefeslioglu et al., 2010; Shaikhina et al., 2019). There is no strict rule for ASM and it has been the subject of many articles (Patidar et al. (2015), Shaheen et al. (2020) and Sagoolmuang and Sinapiromsaran (2020), to name a few). In our study, we prefer to use the "Gini Index" measure (Aruna, 2013) because it is not only easy to implement but also provides a high accuracy score (94%) for the DRP. On the other hand, the performance of the kNN algorithm depends on the value of "k". Therefore, some studies (e.g., Guo et al. (2003) and Song et al. (2007)) consider the problem of choosing the parameter "k". Because the number of features utilized in our training

data is rather small, i.e. 9, the value of "k" is chosen empirically according to the problem at hand (Hassanat et al., 2014); thus, we determine the value of "k" as 5 with the best accuracy scores.

After we obtain a high classification accuracy (e.g. 94% for Decision Tree, and 95% for kNN), we split our training data into two parts (i.e. the training set and the testing set) to get a good generalization for future predictions. Because the training data is randomly split, different classification outputs can diversify the solutions obtained at the end of the learning algorithm. Therefore, we repeat the supervised learning phase 10 times to obtain 10 different classification outputs before implementing our algorithm. Although there are small differences between the objective values of the solutions obtained through each output (e.g., the average value of the solutions for the instances with 30 nodes decreases around 2 percent), we solve the same instance repetitively with these classification outputs to get the best solution on the grounds that this approach does not increase the computational time too much (e.g., the average time for the instances with 30 nodes increases from 0.48s to 3.95s).

The integer solutions that we obtain from the column generation procedure for the training data set show that the order of the nodes in the optimal solutions is almost the same as the order in the TSP solutions. Therefore, we find two different optimal TSP solutions, a tour including the RSs and a tour excluding the RSs for each instance. For the TSP solution with RSs, we exclude the RSs from the solution and obtain an order of the nodes to be processed. For each of the two TSP solutions, we decide which node will be the starting point in the L&F by excluding the arc with the largest distance between two nodes in the related TSP solution. The nodes that have the biggest distance between them give us two starting points. After we decide the starting points, we obtain the giant tours. Hence, we have four different giant tours at the end.

The L&F, whose overview is presented in Figure 2, predicts whether the drone should visit an RS or not whenever the drone is at a node along the giant tour. When it is decided that the drone should visit an RS, the RS causing the minimum increase in the distance of the tour, according to the parameter "Extra Distance", is preferred firstly. If visiting this RS causes infeasibility (i.e., the drone runs out of battery on the way), then the drone visits the nearest RS to it. We start the algorithm at the nearest RS to the starting point, and finish at the nearest RS to the last node that is processed. Once the drone visits all the required nodes, we look for an improvement in the solution. For each simple path which has only one processing node, the algorithm implements a local improvement procedure by adding the node in the simple path to the previous path or the next path in the solution. When the move provides an improvement in the solution, the simple path with one node is excluded from the solution. The algorithm ends after the improvements have been completed. This process is repeated for all giant tours, and the best solution is chosen among them at the end. Algorithm 1 presents the pseudo-code of the L&F. Lines 1 to 6 repeat the L&F procedure by the number of classification outputs and giant tours. Line 8 runs the L&F until all nodes are processed. Lines 9 to 14 set the values of the parameters used in the L&F. Lines 15 to 39 determine the next behavior of the drone by updating the battery charge. Lines 40 to 54 carry out the improvement step. Line 55 returns the best solution found.

Algorithm 1 Pseudo-code of the Learn and Fly algorithm

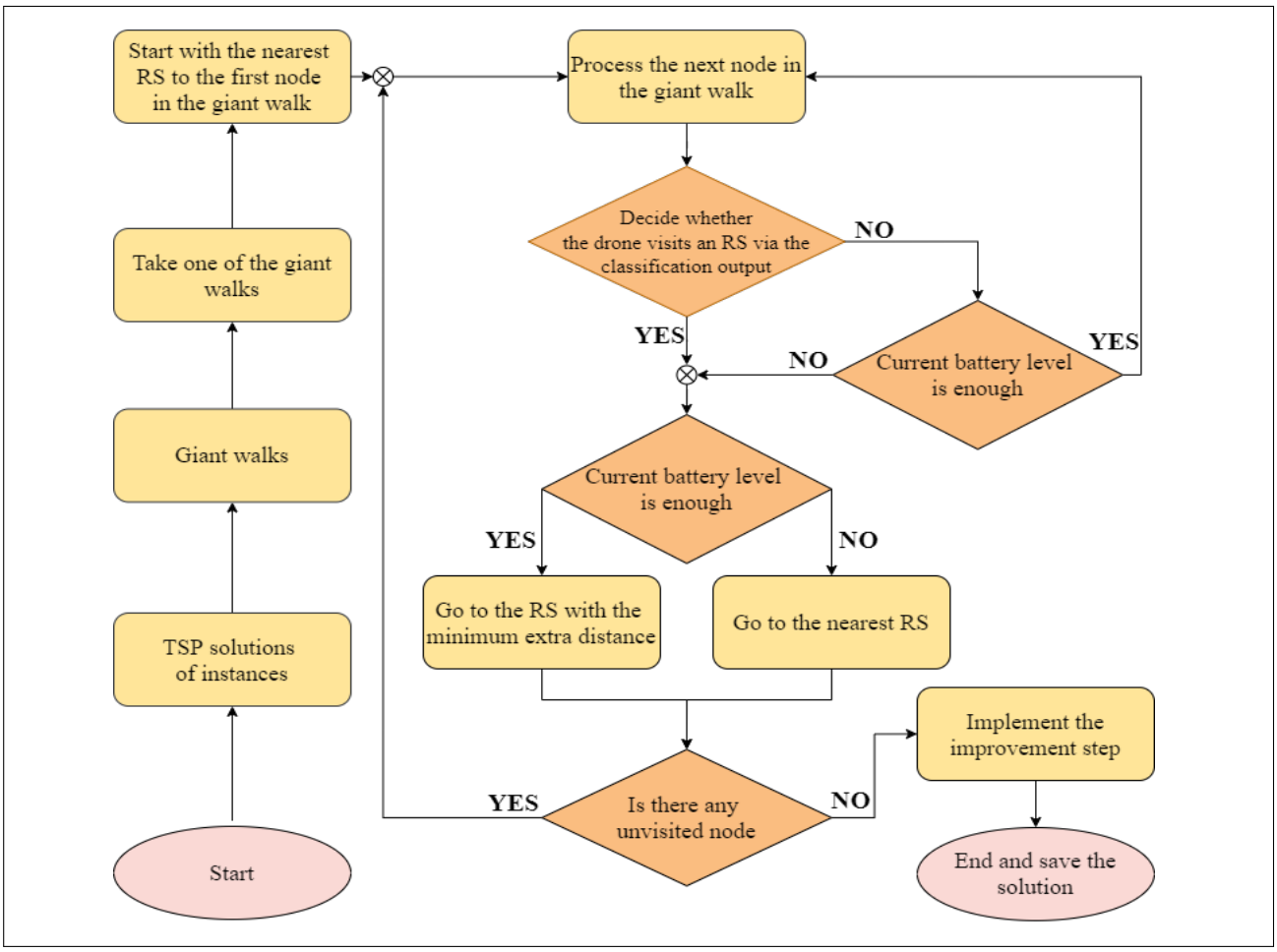
Input: N: Set of nodes that should be processed, R: Set of RSs, τ: Battery Limit, giant tours, classification outputs
Output: A feasible route in which all nodes in N are processed

Begin
$i \leftarrow 1$ to 10 *(the number of the classification outputs)*
for each classification output **do**
  $k \leftarrow 1$ to 4 *(the number of the giant tours)*
  $\tau^- = \tau$
  for each giant tour **do**
    Find the nearest RS to the first node of the giant tour and start routing
    while the number of nodes processed $< |N|$ **do**
      Calculate the values of the features for the current node
      $RS_e \leftarrow$ RS found via feature "Extra Distance"
      $RS_c \leftarrow$ the nearest RS to the current node
      $RS_n \leftarrow$ the nearest RS to the next node in the giant tour
      $d_n \leftarrow$ distance to the next node in the giant tour
      $d_{rs} \leftarrow$ distance between the next node in the giant tour and RS_n
      Predict whether the drone should visit an RS according to the classification output
      if the drone visits an RS **then**
        if distance to $RS_e \leq \tau^-$ **then** *(feasibility check)*
          the current node $\leftarrow RS_e$
          $\tau^- = \tau$
        else
          the current node $\leftarrow RS_c$
          $\tau^- = \tau$
        end if
      else
        if $d_n + d_{rs} \leq \tau^-$ **then** *(feasibility check)*
          the current node $\leftarrow$ the next node in the giant tour
          $\tau^- = \tau^- - d_n$
        else
          if distance to $RS_e \leq \tau^-$ **then** *(feasibility check)*
            the current node $\leftarrow RS_e$
            $\tau^- = \tau$
          else
            the current node $\leftarrow RS_c$
            $\tau^- = \tau$
          end if
        end if
      end if
    end while
    $GiantSol^k \leftarrow$ solution
    Define each group of the nodes from an RS to another RS in the $GiantSol^k$ as a path
    Start improvement steps
    for each path in $GiantSol^k$ **do**
      if there is only one node processed in the path **then**
        if it is feasible and provides an improvement **then**
          Add the node of the previous path or the next path in $GiantSol^k$ whichever provides the best improvement
          $GiantSol^k \leftarrow$ new solution
        end if
      end if
    end for
    $k \leftarrow k + 1$
  end for
  $BestGiantSol^i \leftarrow BestGiantSol^k$
  $i \leftarrow i + 1$
end for
$BestSolution \leftarrow BestGiantSol^i$
End

5. Computational Study

In this section, we carry out experiments to analyze the performance of L&F. Our computational experiments are performed on a computer with Intel(R) Core(TM) i7-7600U CPU @ 2.80 GHz processor

Figure 2: An Overview of the Learn and Fly Algorithm



with a memory of 8 GB. The mathematical models are implemented using Java while Python 3.8.1 is used for the L&F. The mathematical models are solved using ILOG CPLEX 12.8.

5.1. Instance Generation

We test our proposed algorithm on two groups of instances (publicly available at Ermağan et al. (2020)). The first group is targeted towards the precision agriculture application. For each instance, a number of nodes are selected randomly by choosing coordinates in a plane containing some restricted areas. Then, recharge stations are selected as cluster centers of the nodes. The second group of instances are obtained from the "kroA100.tsp" TSP benchmark instance (Krolak et al. (1971)), with an aim to investigate whether the learning outputs obtained from the training set instances that are randomly generated is still useful for solving instances coming from a different network. In both groups, it is assumed that the battery of the drone allows a 30 minute flight time. Accordingly, to ensure that each instance is feasible, we select the nodes in an area with dimensions MxM, where "M" represents the maximum distance that can be covered by the drone with a fully charged battery. The first group

consists of 60 problem instances, having 20, 30, and 40 nodes, where 20 instances are generated for each fixed number of nodes. In all instances, the nodes are selected within an area having the same size so that different node densities can be tested.

In addition to the first instance group, we obtain 20 more instances, each having 40 randomly chosen nodes from the ones in "kroA100.tsp". Since the area in which the nodes are distributed is generally bigger than the MxM area in these instances, we reduce the indicated distances to get feasible solutions with the same battery range as we use in the first group. Because the dimensions of the biggest area that covers all nodes in the instances obtained from 'kroA100.tsp" is almost twice M, we scale the distances to 1/2. Here we want to note that processing a large number of nodes (i.e., more than 40) with a single drone is impractical as it would take too much time and hence multiple drones will need to be used in such cases. Therefore, we limited the size of our test instances by 40 nodes.

To determine the number of RSs, we performed preliminary tests. When we implemented L&F on several instances with different number of RSs, we observed that the drone does not visit one or more RSs at all when the number of RSs is more than five. On the other hand, when the number of the RSs is set between one and four, it causes infeasible solutions for the instances in the MxM area. Therefore, we locate five RSs whose locations are selected as the cluster centers by dividing the nodes into five groups using the K-means++ clustering algorithm (Arthur and Vassilvitskii (2007)) that is solved by Python 3.8.1. After determining the locations of the nodes and the RSs, we calculate the pairwise Euclidean distances and obtain the distance matrix for each instance. We assume that the battery is swapped immediately once the drone lands in a recahrge station, and the time spent to swap the battery equals to 0 (i.e., $t_r = 0$).

5.1.1. Avoiding Restricted Areas

Flying over some regions such as military bases and private property may not be allowed. When the direct flight between two nodes overlaps with a restricted area, the drone has to go around it. Therefore, we update our distance matrices according to the locations and shape of the restricted areas. How this is done is explained in Appendix C.

5.2. Performance Analysis

This subsection presents the results of our computational tests. We first investigate the quality of the integer solutions obtained from the CG. Then, we report the optimality gaps (i.e., the gaps between the lower bound obtained from the CG and the solutions of L&F) of the solutions generated by the L&F algorithm, as well as the run times. We also investigate the features' impact on the accuracy of the classification algorithms. In addition, we implement two rule-based heuristics relying on the most influential features for benchmarking purposes, in order to reveal the advantages of learning.

5.2.1. Quality of the Integer Solutions Obtained from the Column Generation Method

In Table 3, we present the optimality gaps of the integer solutions obtained from CG with respect to the LP relaxation bounds and the computational times for the first group of instances. The average optimality gap obtained from CG is less than 2% for these instances of various sizes. LP solutions

are generated in an average of 8.33 seconds for the instances with 20 nodes. On the other hand, the average computational times are 37.43 seconds and 170.14 seconds for the instances with 30 nodes and 40 nodes, respectively. The integer solutions obtained from CG for the "kroA100.tsp" instances also provide us low optimality gaps. In these instances, while the minimum and maximum optimality gaps are 0% and 4.73%, respectively, the average optimality gap is less than 2% as in the first group of instances. Among the 20 instances obtained from "kroA100.tsp", CG finds the optimal solution in 8 instances. As a result, we conclude that the integer solutions generated by CG provide us a good training set for classification as they are very close to the optimal solutions. We also observe that the computational times of the "kroA100.tsp" instances are well above those of the first group of instances (e.g., the average computational time is 863.51s). The high computational times show that obtaining integer feasible solutions from CG is not a viable method for dynamic environments, where re-optimization should be achieved in very short time.

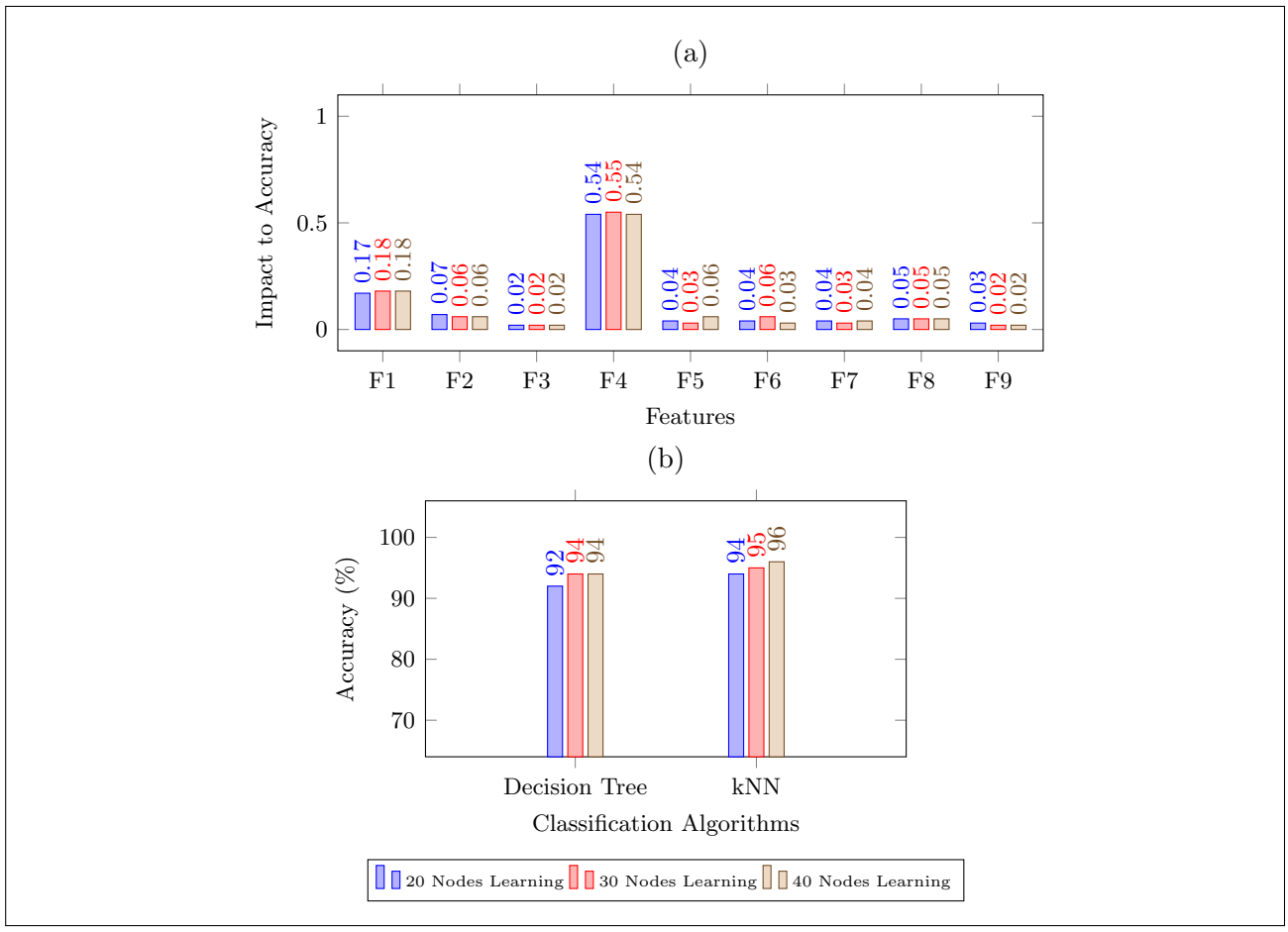
Table 3: Column Generation Results

Instance	**20 Nodes**			**30 Nodes**			**40 Nodes**		
	LP Time(s)	**IP Time(s)**	**GAP (%)**	**LP Time(s)**	**IP Time(s)**	**GAP (%)**	**LP Time(s)**	**IP Time(s)**	**GAP (%)**
1	16.72	16.80	2.80	38.68	38.88	1.03	136.15	136.31	2.88
2	5.30	5.33	0.00	37.00	37.04	0.00	165.29	165.59	2.44
3	7.25	7.28	0.00	32.60	32.64	0.00	180.38	180.48	1.80
4	5.80	5.88	1.20	35.08	35.42	0.33	132.60	133.01	5.31
5	7.47	7.61	0.40	29.48	29.64	0.18	190.06	190.22	1.35
6	6.92	6.97	1.80	26.81	26.93	1.57	137.94	137.98	0.00
7	9.27	9.29	0.00	37.33	37.46	1.24	205.93	206.15	3.20
8	5.88	5.91	0.00	31.28	31.31	0.00	149.39	149.55	1.67
9	8.72	8.87	0.50	21.89	21.92	0.00	114.22	114.39	0.96
10	5.75	5.79	0.00	32.35	32.40	1.67	147.55	147.92	6.65
11	4.79	5.04	0.90	37.10	37.16	0.00	173.39	173.51	1.68
12	12.27	12.30	0.00	33.85	34.02	1.66	340.42	340.63	1.99
13	11.21	11.36	0.20	40.45	40.54	0.73	78.10	78.24	2.43
14	9.38	9.41	0.00	48.27	48.42	1.03	320.02	320.18	0.09
15	9.32	9.40	1.30	29.19	29.34	1.78	130.33	130.44	2.63
16	7.71	7.87	3.40	85.55	85.71	0.78	207.64	207.77	0.57
17	9.56	9.70	1.00	44.38	44.53	1.17	250.35	250.39	0.00
18	9.54	9.57	0.00	30.68	30.83	1.12	137.63	137.73	0.50
19	5.07	5.21	0.00	31.70	31.86	0.19	126.73	126.81	4.35
20	8.61	8.64	0.00	44.87	44.95	0.00	78.59	78.74	0.45
Min:	4.79	5.04	0.00	21.89	21.92	0.00	78.10	78.24	0.00
Max:	16.72	16.80	3.40	85.55	85.71	1.78	340.42	340.63	6.65
Avg:	8.33	8.41	0.68	37.43	37.55	0.72	170.14	170.30	2.05

5.2.2. Performance Analysis of the Classification

In Figure 3a, the features' impact on classification accuracy (Nembrini et al. (2018), Ronaghan (2018)) is demonstrated. Because the feature "Extra Distance" gives the drone a hint about the extra

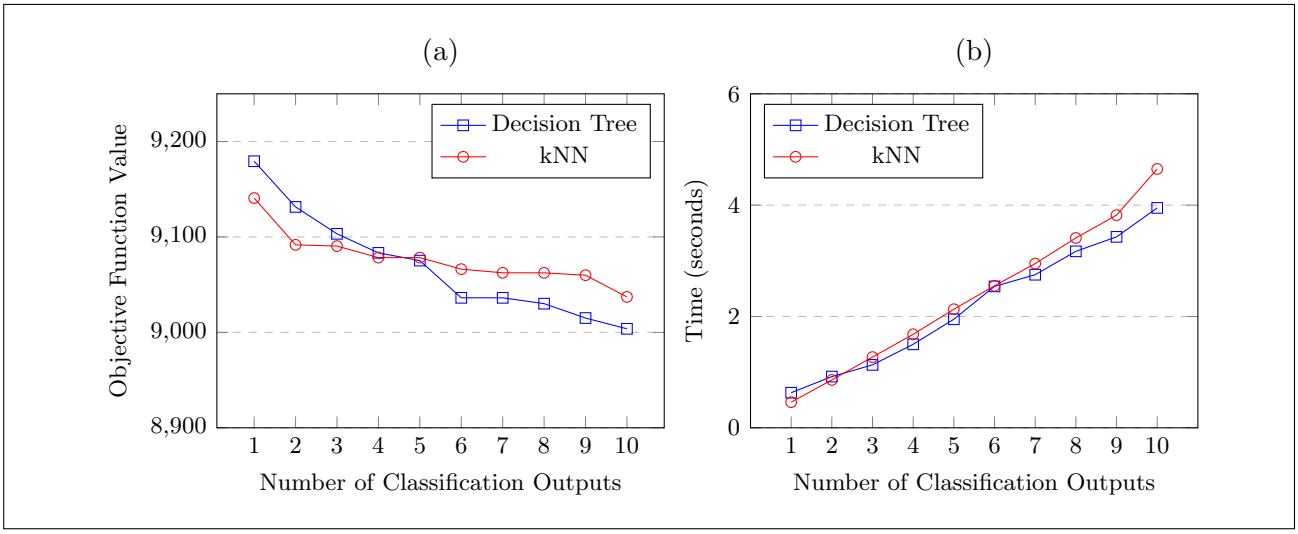
Figure 3: The Analysis of Accuracy Scores



time spent in case an RS is visited, it has the highest importance score compared to others. On the other hand, even though some features do not seem to have an important role in the learning, as seen in Figure 3a, they contribute to the accuracy collectively. The accuracy of learning with only two features leading to the highest impacts, namely, "Battery Level" and "Extra Distance", decreases around 12% (e.g. the accuracy decreases from 94% to 83% for Decision Tree with 30 nodes learning). This result drives us to use more features instead of two, even if the two are the features with by far the highest importance scores.

The final accuracy rates of the classification algorithms are calculated via the k-fold cross validation method which is explained in Subsection 4.2. As seen in Figure 3b, the accuracy scores of the Decision Tree and KNN are around 94% and 95%, respectively. The effect of the number of classification outputs is shown in Figure 4. As explained in Subsection 4.3, the solution improves as the number of classification outputs increases while the increase in the number of the classification outputs does not bring about too much increase in the computational time.

Figure 4: Results with Different Number of Classification Outputs for Instances with 30 Nodes



5.2.3. Performance of the Learn and Fly Algorithm

We report the results of our experiments with the L&F in this subsection. We perform tests on a total of 80 instances, which are generated as explained in Subsection 5.1. We compare the objective value of the solutions found by L&F with the LP relaxation lower bound obtained from the IP model using CG as presented in Section 3.

The L&F algorithm starts by solving a TSP on the set of nodes N. Algorithms developed for the TSP are capable of solving much more larger instances than those required in our application. (Laporte (2010) reviews exact and heuristic solution methods.) For instance, Grötschel and Holland (1991) propose a cutting plane procedure that can solve the symmetric TSP with up to 1000 nodes optimally. Applegate et al. (2009) develop a TSP solver named as Concorde, by which the symmetric TSP can be solved with up to tens of thousand nodes optimally. When we solve our instances with 40 nodes via Concorde (the distance matrices in this study are symmetric), we observe that it obtains the optimal solution within 0.05 seconds. Furthermore, when dynamic environments are considered for the DRP, the solution methods for asymmetric TSP gain importance. (For instance, the wind factor can make the distance matrix asymmetric.) For the asymmetric TSP, Carpaneto et al. (1995) propose a branch and bound algorithm based on the assignment relaxation. In their study, they present the solutions of the instances with up to 2000 vertices that are found in less than three minutes. Hence, if the conditions change while the drone hovers, a new TSP solution that is used at the beginning of the L&F can be found quickly using these algorithms.

In Tables 4-7, the results of our experiments with L&F are reported. In these tables, the columns "Decision Tree Solution" and "kNN Solution" represent the objective values of the L&F solutions obtained when the classifier is Decision Tree and kNN, respectively. The column "Time" shows the execution time of L&F that differs slightly according to the supervised machine learning algorithm

used. The column "LP Relaxation Solution" is used to calculate the values of the columns "Decision Tree GAP(%)" and "kNN GAP(%)" that are the percentage gaps between the L&F and LP relaxation solutions.

In Table 4, for the 20 instances having 20 nodes, the average execution time of L&F is shown as 1.67 seconds for the Decision Tree and as 2.03 seconds for the kNN versions. Besides running fast, the L&F tends to find the solutions very close to the optimal objective value of the model $\mathcal{P}_{\mathcal{DRP}}$ presented in Section 3. The average percentage optimality gaps are 5.78 and 5.76 for the two versions of L&F. L&F finds the optimal solutions for instances numbered as 8, 14, and 20.

Table 4: L&F Results for 20 Node Instances

Instance	**Decision Tree Solution**	**Time (s)**	**kNN Solution**	**Time (s)**	**LP Relaxation Solution**	**Decision Tree GAP(%)**	**kNN GAP(%)**
1	6775.67	1.54	6775.67	2.11	6356.46	6.60	6.60
2	8107.05	1.58	8088.51	2.14	7569.80	7.10	6.85
3	5909.07	1.96	5909.07	1.93	5819.40	1.54	1.54
4	7651.92	1.97	7554.51	1.97	7361.60	3.94	2.62
5	7485.16	1.87	7501.35	1.95	6616.87	13.12	13.37
6	6540.45	1.49	6540.45	2.00	6340.27	3.16	3.16
7	6891.93	1.49	6891.11	2.15	6295.31	9.48	9.46
8	6464.16	1.46	6464.16	1.99	6464.16	0.00	0.00
9	6921.31	1.94	7024.81	2.04	6504.87	6.40	7.99
10	6088.54	1.65	6176.37	1.94	5890.98	3.35	4.84
11	7864.03	1.78	7560.66	2.05	7163.68	9.78	5.54
12	7142.53	1.67	7142.53	1.91	6488.03	10.09	10.09
13	6273.01	1.45	6273.01	2.10	6262.93	0.16	0.16
14	5596.23	1.68	5596.23	1.99	5596.23	0.00	0.00
15	7318.09	1.76	7448.91	2.22	6711.58	9.04	10.99
16	6147.00	1.64	6140.31	1.97	5775.19	6.44	6.32
17	6717.00	1.79	6717.00	2.13	6362.75	5.57	5.57
18	7019.27	1.48	7019.54	2.06	6463.32	8.60	8.61
19	8409.07	1.61	8433.12	1.98	7566.23	11.14	11.46
20	6529.58	1.50	6529.58	1.98	6529.58	0.00	0.00
Min:		1.45		1.91		0.00	0.00
Max:		1.97		2.22		13.12	13.37
Avg:		1.67		2.03		5.78	5.76

When we examine the results of the instances with 30 nodes in Table 5, we see that the average execution time of L&F is still very low (using the Decision Tree, it is 3.95 seconds; using kNN, it is 4.65 seconds). The average gaps are 5.04% and 5.42% for Decision Tree and kNN versions, respectively, which are slightly lower than those values for the 20 node instances. Similarly, in Table 6 in which the results of the instances with 40 nodes are shown, the average execution time for the Decision Tree version of L&F is 5.23 seconds, while it is 6.40 seconds for the kNN version. The average gap values increase by around 1%, along with the increase in the percentage optimality gaps of the integer solutions obtained from CG, that are used to train the classification algorithms.

Table 5: L&F Results for 30 Nodes

Instance	Decision Tree Solution	Time (s)	kNN Solution	Time (s)	LP Relaxation Solution	Decision Tree GAP(%)	kNN GAP(%)
1	8728.81	3.96	8732.50	4.60	8426.53	3.59	3.63
2	8393.88	4.04	8481.64	5.13	8178.18	2.64	3.71
3	9382.27	3.91	9416.26	4.53	8884.83	5.60	5.98
4	9920.40	3.99	9936.79	4.80	9068.23	9.40	9.58
5	9357.51	4.17	9481.22	4.87	8972.68	4.29	5.67
6	8650.90	3.87	8646.01	4.95	8421.15	2.73	2.67
7	9166.98	3.97	9166.98	4.84	8877.82	3.26	3.26
8	9745.99	3.94	9745.99	4.85	9161.35	6.38	6.38
9	9702.83	4.08	9526.65	4.62	8830.52	9.88	7.88
10	8951.49	3.90	9213.71	4.69	8661.48	3.35	6.38
11	9548.58	3.92	9548.58	4.50	8553.15	11.64	11.64
12	7832.98	3.86	7832.98	4.36	7391.22	5.98	5.98
13	8901.86	4.07	8890.67	4.60	8608.82	3.40	3.27
14	10027.19	3.86	10131.85	4.65	9271.73	8.15	9.28
15	9950.94	3.92	10034.50	4.52	9609.29	3.56	4.42
16	8171.58	3.82	8171.58	4.58	7977.91	2.43	2.43
17	7821.38	3.92	7885.10	4.35	7420.97	5.40	6.25
18	8634.67	3.83	8634.67	4.63	8290.79	4.15	4.15
19	8921.25	4.03	8993.07	4.47	8645.94	3.18	4.01
20	8262.94	4.00	8272.25	4.47	8125.88	1.69	1.80
Min:		3.82		4.35		1.69	1.80
Max:		4.17		5.13		11.64	11.64
Avg:		3.95		4.65		5.04	5.42

Next we test L&F on the instances obtained from "kroA100.tsp" benchmark TSP instance and report the results in Table 7. Importantly, to solve these instances with L&F, we use the learning outputs used for instances in Table 6. As seen in Table 7, the average optimality gaps are less than 5%. In addition, L&F runs in a few seconds, while the average execution time of CG is 863.51s. Hence, we reiterate that obtaining integer solutions from the CG is not a viable method in a dynamic environment because of the excessive run time.

Apart from these experiments, we compare L&F with two rule-based heuristic approaches to observe the advantages of learning. These algorithms also utilize an initial TSP tour excluding the RSs, and decide on when to recharge at which RS. The parameters used in the decision rules of the two heuristics are derived from the most influential features of the classification algorithms. In the first algorithm called the Insufficient Battery Heuristic (IBH), the drone visits an RS when the current battery charge is not enough to process the next node in the giant tour. The RS that it visits is chosen according to the parameter "Extra Distance" defined in Table 2. If the RS chosen according to "Extra Distance" causes infeasibility, the drone visits the nearest RS to it as in L&F. The pseudo-code of IBH is presented in Algorithm 2 in Appendix D. The results associated with IBH are reported in the column "Insufficient Battery Heuristic" in Table 8. The second algorithm, whose results are reported in the "Threshold

Table 6: L&F Results for 40 Nodes

Instance	Decision Tree Solution	Time (s)	kNN Solution	Time (s)	LP Relaxation Solution	Decision Tree GAP(%)	kNN GAP(%)
1	10341.69	5.20	10402.52	6.55	9923.85	4.21	4.82
2	10619.56	5.14	10681.03	6.29	10047.38	5.69	6.31
3	11411.97	5.28	11411.97	6.65	10778.16	5.88	5.88
4	11007.66	4.99	11007.66	6.44	10155.77	8.39	8.39
5	10527.50	5.17	10511.15	6.46	10019.35	5.07	4.91
6	10850.51	5.06	11100.78	6.12	10484.35	3.49	5.88
7	9968.22	5.16	9876.44	6.59	9225.15	8.05	7.06
8	10186.18	5.60	10156.79	6.30	9089.24	12.07	11.75
9	9975.37	5.10	10309.88	6.46	9272.06	7.59	11.19
10	10537.71	5.23	10573.52	6.43	10147.30	3.85	4.20
11	10950.36	4.94	10962.03	6.44	10260.86	6.72	6.83
12	10645.97	5.15	10864.64	6.44	9702.62	9.72	11.98
13	11383.12	5.22	11430.49	6.43	10965.81	3.81	4.24
14	9164.47	5.30	9173.45	6.24	8714.25	5.17	5.27
15	10974.73	5.27	11014.47	6.31	9774.68	12.28	12.68
16	10851.90	5.16	10851.90	6.33	10041.09	8.07	8.07
17	10427.37	5.27	10484.45	6.43	9925.46	5.06	5.63
18	11069.05	5.38	11069.05	6.36	10197.18	8.55	8.55
19	9516.11	5.57	9516.11	6.31	8916.56	6.72	6.72
20	10053.25	5.46	10053.25	6.41	9750.66	3.10	3.10
Min:		4.94		6.12		3.10	3.10
Max:		5.60		6.65		12.28	12.68
Avg:		5.23		6.40		6.67	7.17

Heuristic" column of Table 8 is named the Threshold Heuristic (TH). In this method, first the "Battery Level" and "Extra Distance" feature values are found every time the drone visits an RS in the training set solutions. The corresponding averages of these values are set as two thresholds. When the giant tour is processed in the algorithm, if the current battery level and the extra distance that the drone will take by visiting an RS are less than the corresponding threshold values at the same time, the drone visits an RS. The RS that the drone visits is chosen in the same manner as in IBH. The pseudo-code of TH is given in Algorithm 3 in Appendix E. In Table 8 in which the results of the two heuristics for 30 node instances are shown, the average gap of the Insufficient Battery Heuristic is 10.36 percent. Similarly, the average gap of the Threshold Heuristic is 11.87 percent. In comparison, the average gap of L&F for instances with 30 nodes is around 5 percent. The large difference between the performance of L&F and the rule-based heuristics show the importance of using the classification algorithms. Also, it can be observed one more time that the learning accuracy does not depend on only the most important features.

The learning procedure, which is a pre-processing step for the L&F algorithm, takes considerably longer time as the size of the instances grows. However, this can be overcome by using the classification

Table 7: L&F Results for "kroA100" Instances

Instance	Decision Tree Solution	Time (s)	kNN Solution	Time (s)	LP Relaxation Solution	Time (s)	Decision Tree GAP(%)	kNN GAP(%)
1	7854.57	3.60	7854.57	4.99	7607.15	963.91	3.25	3.25
2	8032.88	5.55	8188.60	4.62	7703.13	1036.67	4.28	6.30
3	7682.10	3.58	7682.10	4.27	7396.83	570.91	3.86	3.86
4	7914.71	3.67	7914.71	4.36	7421.44	931.88	6.65	6.65
5	8069.92	3.46	8108.87	4.13	7709.72	706.50	4.67	5.18
6	8202.41	3.24	8204.40	3.90	8010.76	779.18	2.39	2.42
7	7930.33	3.18	8071.04	4.01	7512.94	785.18	5.56	7.43
8	7984.66	3.14	7910.9	4.21	7411.9	585.30	7.73	6.73
9	8110.56	3.12	8110.56	4.42	7850.18	1044.71	3.32	3.32
10	7410.12	3.17	7326.26	4.51	7024.62	658.61	5.49	4.29
11	7671.26	3.21	7617.61	4.03	7379.13	768.55	3.96	3.23
12	8091.78	3.17	8100.59	4.06	7929.77	818.31	2.04	2.15
13	7496.17	3.23	7507.35	4.11	7158.65	726.11	4.71	4.87
14	6952.72	3.18	6943.05	3.81	6797.03	1260.66	2.29	2.15
15	7636.20	3.13	7636.20	3.97	7381.16	1088.11	3.46	3.46
16	8002.84	3.44	7891.47	4.06	7522.98	802.06	6.38	4.90
17	7465.97	3.52	7439.54	4.27	6834.42	1125.58	9.24	8.85
18	7670.84	3.57	7788.55	3.92	7552.96	925.78	1.56	3.12
19	8427.64	4.08	8344.93	4.09	7861.88	1058.10	7.20	6.14
20	8135.38	3.33	8135.38	4.30	7557.32	634.10	7.65	7.65
Min:		3.12		3.81		570.91	1.56	2.15
Max:		5.55		4.99		1260.66	9.24	8.85
Avg:		3.48		4.20		863.51	4.78	4.79

outputs obtained from a smaller-sized set of test instances to solve larger instances. To test the viability of this approach, we utilize the classification outputs from the 20 node instances to solve the instances having 30 and 40 nodes. The results are reported in Table 9. When learning from the 20 node instances, the average gap for the instances with 30 nodes is almost the same as those reported in Table 5. Although the execution time of L&F increases compared with the results in Table 5, it can still be said that the L&F runs very fast because the increase in the execution time is about 2.5 seconds. Similarly, the results of the instances with 40 nodes, where L&F makes its decisions based on the learning outputs from 20 node instances, show that we obtain almost the same quality results as in Table 6. Another observation here is that the execution times do not increase for instances with 40 nodes compared with the results in Table 6. As a result, we assert that learning from instances of smaller size can still lead to high quality solutions in short run times.

6. Conclusions

In our study we propose a learning based fast algorithm, which we name Learn and Fly (L&F), for the DRP that can be utilized in dynamic environments, as well as static ones, by reruns with updated inputs. Our algorithm benefits from machine learning outputs in making key decisions. The supervised machine learning algorithms are appropriate for the DRP because we can classify the target labels as visiting an RS or not visiting an RS. Because we do not have to keep the training data in the memory

Table 8: Results of Insufficient Battery and Threshold Heuristics for 30 Node Instances

Instance	**Insufficient Battery Heuristic**	**Threshold Heuristic**	**LP Relaxation Solutions**	**Insufficient Battery Heuristic GAP(%)**	**Threshold Heuristic GAP(%)**
1	9404.42	9288.72	8426.53	11.6	10.23
2	8589.37	8634.92	8178.18	5.03	5.58
3	10099.3	10099.3	8884.83	13.67	13.67
4	10316.11	10335.74	9068.23	13.76	13.98
5	10025.14	10307.53	8972.68	11.73	14.88
6	9716.17	9943.22	8421.15	15.38	18.07
7	9442.13	9441.98	8877.82	6.36	6.35
8	10147.85	10309.65	9161.35	10.77	12.53
9	10269.66	10179.28	8830.52	16.3	15.27
10	9506.89	10025.24	8661.48	9.76	15.75
11	9986.34	10308.05	8553.15	16.76	20.52
12	8481.7	8477.21	7391.22	14.75	14.69
13	9123.64	9594.3	8608.82	5.98	11.45
14	10683.53	10486.34	9271.73	15.23	13.1
15	10030.22	11162.75	9609.29	4.38	16.17
16	8318.88	8274.59	7977.91	4.27	3.72
17	7982.48	7982.48	7420.97	7.57	7.57
18	9033.27	9033.27	8290.79	8.96	8.96
19	9205.3	9205.3	8645.94	6.47	6.47
20	8812.02	8812.02	8125.88	8.44	8.44
			Min:	4.27	3.72
			Max:	16.76	20.52
			Avg:	10.36	11.87

for solving new instances, the learning phase does not cause an increase in computation time each time the algorithm is rerun. The learning output mapping our features to predict the labels is enough to execute the further steps of our learning based algorithm.

A critical issue in the learning phase is which features are used. We define 9 features according to the parameters playing a foremost role in the timing of recharging. Clearly, the training data set affects the quality of the resulting solutions. In our study, we apply a column generation procedure on a novel integer program with path-segment and cycle variables, so as to obtain integer feasible solutions for the DRP that are very close to the optimal solutions (within 2 percent on the average) as the training set. L&F takes advantage of two supervised learning algorithms, as alternatives to each other, for classification.

We show that L&F provides near-optimal solutions in our computational tests with planar data having 20, 30 and 40 nodes, as well as other 40 node instances derived from a TSP benchmark instance. The successful performance of L&F is confirmed, even if the training solutions come from a smaller sized data set. Benchmarking with two rule-based algorithms also verified the good performance of L&F. Since the L&F algorithm runs in a few seconds, it can be used in dynamic environments by rerunning the algorithm as the input parameters of DRP change.

For future research, different approaches may be employed for the learning phase to improve the optimality gaps further. The features that we proposed and their impact on the classification accuracy rate can provide hints for future algorithms. Also, it remains a challenge to apply the ideas presented in this study to a multi-drone routing problem that can be solved in dynamic environments.

Table 9: L&F Results with Learning from 20 Node Instances

	30 NODES							40 NODES						
Instance	**Decision Tree Solution**	**Time (s)**	**kNN Solution**	**Time (s)**	**LP Relaxation Solution**	**Decision Tree GAP (%)**	**kNN GAP (%)**	**Decision Tree Solution**	**Time (s)**	**kNN Solution**	**Time (s)**	**LP Relaxation Solution**	**Decision Tree GAP (%)**	**kNN GAP (%)**
1	8732.50	6.61	8796.67	7.70	8426.53	3.63	4.39	10402.52	5.27	10341.69	6.77	9923.85	4.82	4.21
2	8543.22	6.09	8545.38	7.59	8178.18	4.46	4.49	10514.29	5.60	10835.28	6.49	10047.38	4.65	7.84
3	9374.36	6.15	9416.26	7.28	8884.83	5.51	5.98	11411.97	5.87	11411.97	6.60	10778.16	5.88	5.88
4	9936.79	6.52	9999.88	7.03	9068.23	9.58	10.27	10995.43	5.39	10995.43	6.39	10155.77	8.27	8.27
5	9442.14	6.57	9667.28	7.36	8972.68	5.23	7.74	10630.99	5.61	10574.71	6.92	10019.35	6.10	5.54
6	8650.90	6.46	8650.90	7.35	8421.15	2.73	2.73	10982.09	5.06	10714.27	6.95	10484.35	4.75	2.19
7	9166.98	6.38	9166.98	7.34	8877.82	3.26	3.26	10010.96	5.22	10311.27	6.45	9225.15	8.52	11.77
8	9719.83	6.54	9745.99	7.56	9161.35	6.10	6.38	10107.39	5.85	10199.08	6.58	9089.24	11.20	12.21
9	9677.84	6.56	9702.83	7.20	8830.52	9.60	9.88	9997.83	5.56	10219.13	6.50	9272.06	7.83	10.21
10	9121.96	6.42	9383.28	7.16	8661.48	5.32	8.33	10573.52	5.24	10578.86	6.53	10147.30	4.20	4.25
11	9655.94	6.23	9660.34	7.33	8553.15	12.89	12.94	10910.05	5.59	10962.03	6.38	10260.86	6.33	6.83
12	7832.98	6.18	7833.63	7.14	7391.22	5.98	5.99	10877.60	5.45	10911.23	6.24	9702.62	12.11	12.46
13	8901.86	6.41	8852.64	7.35	8608.82	3.40	2.83	11434.24	5.16	11441.96	6.60	10965.81	4.27	4.34
14	10349.47	6.43	10436.16	7.38	9271.73	11.62	12.56	9153.59	4.92	9133.73	6.52	8714.25	5.04	4.81
15	9950.94	6.12	10034.50	7.45	9609.29	3.56	4.42	11146.88	5.49	11202.39	6.65	9774.68	14.04	14.61
16	8173.40	6.44	8171.58	7.57	7977.91	2.45	2.43	10892.05	5.24	10851.90	6.59	10041.09	8.47	8.07
17	7821.38	6.34	7884.36	9.56	7420.97	5.40	6.24	10457.22	5.50	10722.72	6.88	9925.46	5.36	8.03
18	8634.67	6.32	8634.67	7.78	8290.79	4.15	4.15	10992.31	5.38	11069.05	6.50	10197.18	7.80	8.55
19	8921.25	6.63	8993.07	7.49	8645.94	3.18	4.01	9724.25	5.02	9516.11	6.25	8916.56	9.06	6.72
20	8272.25	6.51	8262.94	7.14	8125.88	1.80	1.69	10053.25	5.16	10053.25	6.37	9750.66	3.10	3.10
Min.:		**6.09**		**7.03**		**1.80**	**1.69**	**Min.:**	**4.92**		**6.24**		**3.10**	**2.19**
Max.:		**6.63**		**9.56**		**12.89**	**12.94**	**Max.:**	**5.87**		**6.95**		**14.04**	**14.61**
Avg.:		**6.40**		**7.49**		**5.49**	**6.04**	**Avg.:**	**5.38**		**6.56**		**7.09**	**7.49**

CRediT authorship contribution statement

Umut Ermağan: Conceptualization, Methodology, Software, Writing-original draft. **Barış Yıldız:** Conceptualization, Methodology, Software, Writing-review & editing, Supervision. **F. Sibel Salman:** Conceptualization, Methodology, Writing-review & editing, Supervision, Project administration.

Declaration of Competing Interest

The authors declare that they have no known competing financial interests or personal relationships that could have appearedto influence the work reported in this paper.

Appendix A. Formulations of the Pricing Problems

With the parameters and the decision variables summarized in Table A.10, we present the formulations of the two pricing problems.

Table A.10: Notation

Sets	**Description**
N_{od}	$N \cup \{o, d\}$ for all $o \in R, d \in R \setminus \{o\}$
N_o	N $\cup$ $\{o\}$ for all $o \in R$
A_{od}	Set of arcs in A for all $o \in R, d \in R \setminus \{o\}$ where o and d are the starting and ending points of the path, respectively
A_o	Set of arcs in A for all $o \in R$ where o is the starting and ending point of the cycle
Parameters	**Description**
$\bar{t}_{ij}/\hat{t}_{ij}$	Cost of arc (i, j)
t_{ij}	Resource usage of arc (i, j)
τ	Total flight time of the drone with fully charged batteries.
Decision Variables	**Description**
u_{ij}	Binary variable that indicates whether arc (i, j) is used
v_i	Binary variable that indicates whether node i is in the path/cycle

Pricing Problem for the Path Variables: To find simple paths with negative reduced costs, for each pair $(o, d) \in R \times R$ where $o \neq d$ we solve the pricing problem formulated as follows:

$$\min \sum_{(i,j)\in A_{od}} \bar{t}_{ij} u_{ij} \tag{A.1}$$

s.t.

$$\sum_{(ij)\in A_{od}} u_{ij} - \sum_{(ji)\in A_{od}} u_{ji} = \begin{cases} 1 & \text{if } i = o \\ -1 & \text{if } i = d \\ 0 & \text{ow.} \end{cases} \qquad \forall i \in N_{od} \tag{A.2}$$

$$\sum_{(ij)\in A_{od}} u_{ij} = v_i \qquad \forall i \in N_{od} \tag{A.3}$$

$$\sum_{(ij)\in A_{od}} u_{ij} = \sum_{i\in N_{od}} v_i - 1 \tag{A.4}$$

$$\sum_{(ij)\in A_{od}} t_{ij}u_{ij} \le \tau \tag{A.5}$$

$$\sum_{(ij)\in S(A_{od})} u_{ij} \le \sum_{i\in S\setminus\{k\}} v_i \qquad \forall S \subset N_{od}, k \in S \tag{A.6}$$

$$u_{ij} \in \{0,1\} \qquad \forall (ij) \in A_{od} \tag{A.7}$$

$$v_i \in \{0,1\} \qquad \forall i \in N_{od} \tag{A.8}$$

Pricing Problem for the Cycle Variables: Similarly, to find simple cycles with negative reduced costs, for each recharge station $o \in R$ we solve the pricing problem formulated as follows:

$$\min \sum_{(i,j)\in A_o} \hat{t}_{ij}u_{ij} \tag{A.9}$$

s.t.

$$\sum_{(ij)\in A_o} u_{ij} - \sum_{(ji)\in A_o} u_{ji} = \begin{cases} 1 & \text{if } i = o \\ -1 & \text{if } i = \hat{o} \\ 0 & \text{ow.} \end{cases} \qquad \forall i \in N_o \tag{A.10}$$

$$\sum_{(ij)\in A_o} u_{ij} = v_i \qquad \forall i \in N_o \tag{A.11}$$

$$\sum_{(ij)\in A_o} u_{ij} = \sum_{i\in N_o} v_i - 1 \tag{A.12}$$

$$\sum_{(ij)\in A_o} t_{ij}u_{ij} \le \tau \tag{A.13}$$

$$\sum_{(ij)\in S(A_o)} u_{ij} \le \sum_{i\in S\setminus\{k\}} v_i \qquad \forall S \subset N_o, k \in S \tag{A.14}$$

$$u_{ij} \in \{0,1\} \qquad \forall (ij) \in A_o \tag{A.15}$$

$$v_i \in \{0,1\} \qquad \forall i \in N_o \tag{A.16}$$

The objective functions (A.1) and (A.9) minimize the total arc costs of the path and the cycle, respectively. If a node is the origin or destination point in the path/cycle, it should have degree one. Otherwise, its degree should be two. This is forced by the flow balance constraints (A.2) and (A.10). Constraints (A.3), (A.4) for the paths and (A.11), (A.12) for the cycles ensure that the solution is a simple path. Constraints (A.6) and (A.14) eliminate the sub-tours formed by arcs. Constraints (A.5)/(A.13) prevent the paths/cycles whose costs are larger than the battery limit from being added to the restricted master problem. Finally, constraints (A.15)-(A.16) are the variable domain restrictions.

Branch and Cut Algorithm for the Pricing Problem: We use the branch and cut algorithm to solve the pricing problems because they may be comprised of exponential numbers of subtour

elimination constraints. In the branch and cut tree, the LP relaxations of the pricing problems are solved at each node. If the solution that is found is integer and it is not feasible at any node, i.e. there are sub-tours, we implement the lazy constraints call-back to eliminate the sub-tours. Otherwise, we branch. In the algorithm, the branching strategies of CPLEX are used to explore the branch and cut tree.

Appendix B. Formulation of the DRP on the Giant Tour Network

Table B.11: Notation Used in the Formulation of the DRP on the Giant Tour Network

Sets	**Description**
V	Set of nodes consisting of nodes that are required to be visited and RSs
A	Set of arcs
N	Set of nodes that are required to be visited
R	Set of RSs
S	Subsets of the set V
$S(A)$	The set of arcs of the subset S
A_{tour}	Set of arcs consisting of the arcs of the giant tour and $\|N\|x\|R\|$
Parameters	**Description**
t_{ij}	Travel time associated with arc (i,j)
τ	Total flight time of the drone with fully charged battery
M	Big number
Decision Variables	**Description**
f_{ij}	Binary variable that indicates whether arc $(i,j) \in A_{tour}$ is used
o_r	Binary variable that indicates whether the drone starts its tour at $r \in R$
d_r	Binary variable that indicates whether the drone ends its tour at $r \in R$
n_i	Binary variable that indicates whether the drone visits node $i \in N$
v_i	Battery charge level when the drone leaves node $i \in V$

$$\min \sum_{(i,j)\in A_{tour}} t_{ij} f_{ij} \tag{B.1}$$

s.t.

$$\sum_{r\in R} o_r = 1 \tag{B.2}$$

$$\sum_{r\in R} d_r = 1 \tag{B.3}$$

$$d_r - o_r + \sum_{(i,j)\in A_{tour}:i=r} f_{ij} - \sum_{(j,i)\in A_{tour}:i=r} f_{ji} = 0 \qquad \forall r \in R \tag{B.4}$$

$$\sum_{j:(i,j)\in A_{tour}} f_{ij} - \sum_{j:(j,i)\in A_{tour}} f_{ji} = 0 \qquad \forall i \in N \tag{B.5}$$

$$\sum_{j:(i,j)\in A_{tour}} f_{ij} = n_i \qquad \forall i \in N \tag{B.6}$$

$$\sum_{i \in N} n_i = |N| \tag{B.7}$$

$$\sum_{j:(r,j) \in A_{tour}} f_{rj} + \sum_{j:(j,r) \in A_{tour}} f_{jr} \geq 1 \qquad \forall r \in R \tag{B.8}$$

$$v_j \leq v_i - t_{ij} f_{ij} + M(1 - f_{ij}) \qquad \forall (i,j) \in A_{tour} : i \in N \tag{B.9}$$

$$v_j \leq \tau - t_{ij} f_{ij} \qquad \forall (i,j) \in A_{tour} : i \in R \tag{B.10}$$

$$f_{ir} \leq \sum_{j:(r,j) \in A_{tour}} f_{rj} \qquad \forall i \in N, r \in R : (i,r) \in A_{tour} \tag{B.11}$$

$$\sum_{(i,j) \in S(A): j \notin S} f_{ij} + \sum_{(ij) \in S(A): i \notin S} f_{ij} \geq 1 \qquad \forall S \subset V : S(A) \subset A_{tour} \tag{B.12}$$

$$v_i \geq 0 \qquad \forall i \in V \tag{B.13}$$

$$f_{ij} \in \{0,1\} \qquad \forall (i,j) \in A_{tour} \tag{B.14}$$

$$n_i \in \{0,1\} \qquad \forall i \in N \tag{B.15}$$

$$o_r \in \{0,1\} \qquad \forall r \in R \tag{B.16}$$

$$d_r \in \{0,1\} \qquad \forall r \in R \tag{B.17}$$

The objective function (B.1) minimizes the total travel time of the drone. Constraints (B.2) and (B.3) determine the start and end nodes of the drone's tour. Constraints (B.4) and (B.5) are the flow balance constraints for the RSs and the nodes to be processed, respectively. These constraints ensure that the number of incoming arcs to a node or an RS is the same with the number of outgoing arcs. Constraints (B.6) and (B.7) ensure every node to be processed. Constraints (B.8) ensure every RS to be used. Constraints (B.9) and (B.10) are related to the current battery charge of the drone. They provide that the battery charge of the drone is greater than 0 and less than the battery charge at the previous processed node. Constraints (B.11) ensure that the drone follows the order in the giant tour. Constraints (B.12) eliminate the subtour formed by arcs. To achieve this, they ensure that at least one incoming or outgoing arc which does not belong to the subset S is selected in the solution. Finally, constraints (B.13)-(B.17) are the variable domain restrictions.

Appendix C. Updating the Distance Matrix

In order to detect which arcs in the complete directed graph intersect with restricted areas, we divide the lines between each pair of nodes into multiple points. Then, we check whether the line passes through the restricted area or not by using two equations from analytical geometry that are specified in the following.

We define all the restricted areas as a polygon, where A represents its area and (x, y) represents the coordinates of the corners. The formula of the area is as follows:

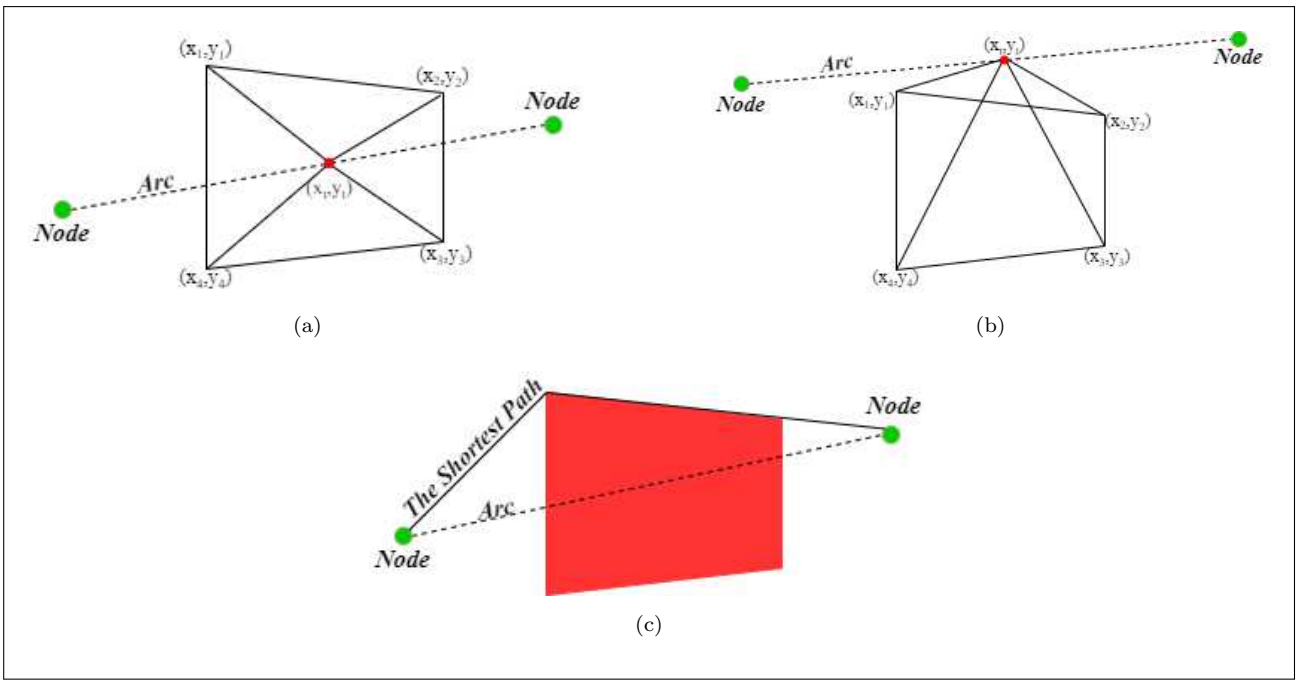
$$A = \tfrac{1}{2}|(x_1 y_2 - y_1 x_2) + (x_2 y_3 - y_2 x_3) + ... + (x_n y_1 - y_n x_1)|$$

With the points which we get from the lines, we draw four triangles whose corners consist of the point (x_t, y_t) we found and two corners of the polygon as shown in Figure C.5a and Figure C.5b. Each triangle's area can be found as follows:

$$A_{triangle} = \frac{1}{2}|x_t y_1 + x_1 y_2 + x_2 y_t - (x_1 y_t + x_2 y_1 + x_t y_2)|$$

By using these two formulas, we check all edges and determine which edges intersect with the restricted areas. For any point on the arc, if the sum of the areas of the four triangles equals to the area of the quadrilateral, this implies that the point is inside the quadrilateral as in Figure C.5a. On the other hand, if the sum of the areas of the four triangles is greater than the area of the quadrilateral for every point on the arc, we can deduce that the arc does not intersect with the restricted area. After

Figure C.5: Avoiding Restricted Areas



detecting the arcs which intersect with the restricted area, the distances of the shortest paths between all pairs of the nodes associated with these arcs are calculated and the distance matrix is updated accordingly. An example of an arc that intersects with a restricted area and the shortest path between the related nodes is provided in Figure C.5c.

Appendix D. Insufficient Battery Heuristic

Algorithm 2 Pseudo-code of the Insufficient Battery Algorithm

Input: V: Set of nodes, N: Set of nodes that should be processed, R: Set of RSs, τ: Battery Limit, giant tours
Output: A feasible route in which all nodes in N are processed

Begin
$k \leftarrow 1$ to 4 *(the number of giant tours)*
$\tau^- = \tau$
for each giant tour **do**
  Find the nearest RS to the first node of the giant tour, and start routing
  while the number of nodes processed $< |N|$ **do**
    Calculate the values of the parameters for the current node
    $RS_e \leftarrow$ RS found via the Extra Distance feature for the current node
    $RS_n \leftarrow$ the RS found via the Extra Distance feature for the next node in the giant tour
    $d_n \leftarrow$ distance to the next node in the giant tour
    $d_{rs} \leftarrow$ distance between the next node in the giant tour and RS_n
    if $d_n + d_{rs} \leq \tau^-$ **then** *(feasibility check)*
      the current node $\leftarrow$ the next node in the giant tours
      $\tau^- = \tau^- - d_n$
    else
      the current node $\leftarrow RS_e$
      $\tau^- = \tau$
    end if
  end while
  $GiantSol^k \leftarrow$ solution
  Define each group of the nodes from a RS to another RS in the $GiantSol^k$ as a path
  Start improvement steps
  for each path in $GiantSol^k$ **do**
    if there is only one node processed in the path **then**
      if it is feasible and provides an improvement **then**
        Add the node the previous path or the next path in $GiantSol^k$ whichever provides the best improvement
        $GiantSol^k \leftarrow$ new solution
      end if
    end if
  end for
  $k \leftarrow k + 1$
end for
$BestSolution \leftarrow BestGiantSol^k$
End

Appendix E. The Threshold Heuristic

Algorithm 3 Pseudo-code of the Threshold Heuristic

Input: V: Set of nodes, N: Set of nodes that should be processed, R: Set of RSs, τ: Battery Limit, giant tours, the threshold parameters (B_{TH}: Battery Level and D_{TH}:Extra Distance)
Output: A feasible route in which all nodes in N are processed

Begin
$k \leftarrow 1$ to 4 *(the number of giant tours)*
$\tau^- = \tau$
for each giant tour **do**
  Find the nearest RS to the first node of the giant tour, and start routing
  while the number of nodes processed $< |N|$ **do**
    Calculate the values of the parameters for the current node
    $RS_e \leftarrow$ RS found via the "Extra Distance" feature
    $RS_c \leftarrow$ the nearest RS to the current node
    $RS_n \leftarrow$ the nearest RS to the next node in the giant tour
    $d_e \leftarrow$ the extra distance
    $d_n \leftarrow$ distance to the next node in the giant tour
    $d_{rs} \leftarrow$ distance between the next node in the giant tour and RS_n
    Decide whether the drone should visit an RS according to the threshold parameters
    if $\tau^- \leq B_{TH}$ and $d_e \leq D_{TH}$ **then**
      The drone should visit an RS
      if distance to $RS_e \leq \tau^-$ **then** *(feasibility check)*
        the current node $\leftarrow RS_e$
        $\tau^- = \tau$
      else
        the current node $\leftarrow RS_c$
        $\tau^- = \tau$
      end if
    else
      if $d_n + d_{rs} \leq \tau^-$ **then** *(feasibility check)*
        the current node $\leftarrow$ the next node in the giant tour
        $\tau^- = \tau^- - d_n$
      else
        if distance to $RS_e \leq \tau^-$ **then** *(feasibility check)*
          the current node $\leftarrow RS_e$
          $\tau^- = \tau$
        else
          current node $\leftarrow RS_c$
          $\tau^- = \tau$
        end if
      end if
    end if
  end while
  $GiantSol^k \leftarrow$ solution
  Define each group of the nodes from an RS to another RS in the $GiantSol^k$ as a path
  Start improvement steps
  for each path in $GiantSol^k$ **do**
    if there is only one node processed in the path **then**
      if it is feasible and provides an improvement **then**
        Add the node the previous path or the next path in $GiantSol^k$ whichever provides the best improvement
        $GiantSol^k \leftarrow$ new solution
      end if
    end if
  end for
  $k \leftarrow k + 1$
end for
$BestSolution \leftarrow BestGiantSol^k$
End

References

Al-Sabban, W. H., Gonzalez, L. F. and Smith, R. N. (2013), Wind-energy based path planning for unmanned aerial vehicles using markov decision processes, *in* '2013 IEEE International Conference on Robotics and Automation', IEEE, pp. 784–789.

Applegate, D. L., Bixby, R. E., Chvátal, V., Cook, W., Espinoza, D. G., Goycoolea, M. and Helsgaun, K. (2009), 'Certification of an optimal tsp tour through 85,900 cities', *Operations Research Letters* **37**(1), 11–15.

Arthur, D. and Vassilvitskii, S. (2007), k-means++: the advantages of careful seeding, *in* 'SODA'07: proceedings of the eighteenth annual ACM-SIAM symposium on discrete algorithms. Society for Industrial and Applied Mathematics, Philadelphia, PA', pp. 1027–1035.

Aruna, R. (2013), 'Construction of decision tree : Attribute selection measures', *International Journal of Advancements in Research Technology* **2**(4), 343–346.

Barrientos, A., Colorado, J., Cerro, J. d., Martinez, A., Rossi, C., Sanz, D. and Valente, J. (2011), 'Aerial remote sensing in agriculture: A practical approach to area coverage and path planning for fleets of mini aerial robots', *Journal of Field Robotics* **28**(5), 667–689.

Beasley, J. E. and Christofides, N. (1989), 'An algorithm for the resource constrained shortest path problem', *Networks* **19**(4), 379–394.

Behnke, M., Kirschstein, T. and Bierwirth, C. (2020), 'A column generation approach for an emission-oriented vehicle routing problem on a multigraph', *European Journal of Operational Research* **288**(3), 794–809.

Carpaneto, G., Dell'Amico, M. and Toth, P. (1995), 'Exact solution of large-scale, asymmetric traveling salesman problems', *ACM Transactions on Mathematical Software (TOMS)* **21**(4), 394–409.

Chow, J. Y. (2016), 'Dynamic uav-based traffic monitoring under uncertainty as a stochastic arc-inventory routing policy', *International Journal of Transportation Science and Technology* **5**(3), 167–185.

Crainic, T. G., Gendreau, M. and Potvin, J.-Y. (2009), 'Intelligent freight-transportation systems: Assessment and the contribution of operations research', *Transportation Research Part C: Emerging Technologies* **17**(6), 541–557.

Cui, J.-H., Wei, R.-X., Liu, Z.-C. and Zhou, K. (2018), 'Uav motion strategies in uncertain dynamic environments: A path planning method based on q-learning strategy', *Applied Sciences* **8**(11), 2169.

Desaulniers, G., Errico, F., Irnich, S. and Schneider, M. (2016), 'Exact algorithms for electric vehicle-routing problems with time windows', *Operations Research* **64**(6), 1388–1405.

Dorigo, M., Maniezzo, V. and Colorni, A. (1996), 'Ant system: optimization by a colony of cooperating agents', *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)* **26**(1), 29–41.

Erdelić, T. and Carić, T. (2019), 'A survey on the electric vehicle routing problem: variants and solution approaches', *Journal of Advanced Transportation* **2019**.

Ermağan, U., Yıldız, B. and Salman, S. (2020), 'Datasets for drone routing problem, v2', *Mendeley Data* . doi: 10.17632/jxwsp9ng5c.2.

Focacci, F., Lodi, A. and Milano, M. (2002), 'A hybrid exact algorithm for the tsptw', *INFORMS Journal on Computing* **14**(4), 403–417.

Gebbers, R. and Adamchuk, V. I. (2010), 'Precision agriculture and food security', *Science* **327**(5967), 828–831.

Goeke, D. and Schneider, M. (2015), 'Routing a mixed fleet of electric and conventional vehicles', *European Journal of Operational Research* **245**(1), 81 – 99.

Grötschel, M. and Holland, O. (1991), 'Solution of large-scale symmetric travelling salesman problems', *Mathematical Programming* **51**(1-3), 141–202.

Guerriero, F., Surace, R., Loscra, V. and Natalizio, E. (2014), 'A multi-objective approach for unmanned aerial vehicle routing problem with soft time windows constraints', *Applied Mathematical Modelling* **38**(3), 839 – 852.

Guo, G., Wang, H., Bell, D., Bi, Y. and Greer, K. (2003), Knn model-based approach in classification, *in* 'OTM Confederated International Conferences" On the Move to Meaningful Internet Systems"', Springer, pp. 986–996.

Hassanat, A. B., Abbadi, M. A., Altarawneh, G. A. and Alhasanat, A. A. (2014), 'Solving the problem of the k parameter in the knn classifier using an ensemble learning approach', *International Journal of Computer Science and Information Security* **12**(8), 33–39.

Hiermann, G., Hartl, R. F., Puchinger, J. and Vidal, T. (2019), 'Routing a mix of conventional, plug-in hybrid, and electric vehicles', *European Journal of Operational Research* **272**(1), 235 – 248.

Khoufi, I., Laouiti, A. and Adjih, C. (2019), 'A survey of recent extended variants of the traveling salesman and vehicle routing problems for unmanned aerial vehicles', *Drones* **3**(3), 66.

Kim, S. J., Lim, G. J. and Cho, J. (2018), 'Drone flight scheduling under uncertainty on battery duration and air temperature', *Computers & Industrial Engineering* **117**, 291–302.

Kohavi, R. (1995), A study of cross-validation and bootstrap for accuracy estimation and model selection, *in* 'Ijcai', Vol. 14, no.2, Montreal, Canada, pp. 1137–1145.

Kotsiantis, S. B., Zaharakis, I. and Pintelas, P. (2007), 'Supervised machine learning: A review of classification techniques', *Emerging Artificial Intelligence Applications in Computer Engineering* **160**(1), 3–24.

Krolak, P., Felts, W. and Marble, G. (1971), 'A man-machine approach toward solving the traveling salesman problem', *Communications of the ACM* **14**(5), 327–334.

Laporte, G. (2010), 'A concise guide to the traveling salesman problem', *Journal of the Operational Research Society* **61**(1), 35–40.

Lee, Y., Chiu, S. Y. and Ryan, J. (1996), 'A branch and cut algorithm for a steiner tree-star problem', *INFORMS Journal on Computing* **8**(3), 194–201.

Li, M., Zhen, L., Wang, S., Lv, W. and Qu, X. (2018), 'Unmanned aerial vehicle scheduling problem for traffic monitoring', *Computers & Industrial Engineering* **122**, 15 – 23.

Liao, C.-S., Lu, S.-H. and Shen, Z.-J. (2016), 'The electric vehicle touring problem', *Transportation Research Part B: Methodological* **86**, 163 – 180.

Modaresi, S., Sauré, D. and Vielma, J. P. (2020), 'Learning in combinatorial optimization: What and how to explore', *Operations Research* **to appear**.

Nefeslioglu, H., Sezer, E., Gokceoglu, C., Bozkir, A. and Duman, T. (2010), 'Assessment of landslide susceptibility by decision trees in the metropolitan area of istanbul, turkey', *Mathematical Problems in Engineering* **2010**.

Nembrini, S., König, I. R. and Wright, M. N. (2018), 'The revival of the gini importance?', *Bioinformatics* **34**(21), 3711–3718.

Otto, A., Agatz, N., Campbell, J., Golden, B. and Pesch, E. (2018), 'Optimization approaches for civil applications of unmanned aerial vehicles (uavs) or aerial drones: A survey', *Networks* **72**(4), 411–458.

Patidar, P., Dangra, J. and Rawar, M. (2015), 'Decision tree c4. 5 algorithm and its enhanced approach for educational data mining', *Engineering Universe for Scientific Research and Management* **7**(2), 1–14.

Ronaghan, S. (2018), 'The mathematics of decision trees, random forest and feature importance in scikit-learn and spark', *Medium* **May, 11**.

Sagoolmuang, A. and Sinapiromsaran, K. (2020), 'Decision tree algorithm with class overlapping-balancing entropy for class imbalanced problem', *International Journal of Machine Learning and Computing* **10**(3), 444–451.

Sassi, O. and Oulamara, A. (2017), 'Electric vehicle scheduling and optimal charging problem: complexity, exact and heuristic approaches', *International Journal of Production Research* **55**(2), 519–535.

Schneider, M., Stenger, A. and Goeke, D. (2014), 'The electric vehicle-routing problem with time windows and recharging stations', *Transportation Science* **48**(4), 500 – 520.

Shaheen, M., Zafar, T. and Ali Khan, S. (2020), 'Decision tree classification: Ranking journals using igidi', *Journal of Information Science* **46**(3), 325–339.

Shaikhina, T., Lowe, D., Daga, S., Briggs, D., Higgins, R. and Khovanova, N. (2019), 'Decision tree and random forest models for outcome prediction in antibody incompatible kidney transplantation', *Biomedical Signal Processing and Control* **52**, 456–462.

Song, Y., Huang, J., Zhou, D., Zha, H. and Giles, C. L. (2007), Iknn: Informative k-nearest neighbor pattern classification, *in* 'European Conference on Principles of Data Mining and Knowledge Discovery', Springer, pp. 248–264.

Sutton, R. S. and Barto, A. G. (2018), *Reinforcement learning: An introduction*, MIT press.

Swain, K. C., Jayasuriya, H. P. and Salokhe, V. M. (2007), 'Suitability of low-altitude remote sensing images for estimating nitrogen treatment variations in rice cropping for precision agriculture adoption', *Journal of Applied Remote Sensing* **1**(1), 013547.

Thibbotuwawa, A., Bocewicz, G., Radzki, G., Nielsen, P. and Banaszak, Z. (2020), 'Uav mission planning resistant to weather uncertainty', *Sensors* **20**(2), 515.

Watkins, C. J. and Dayan, P. (1992), 'Q-learning', *Machine learning* **8**(3-4), 279–292.

Yıldız, B., Arslan, O. and Karaşan, O. E. (2016), 'A branch and price approach for routing and refueling station location model', *European Journal of Operational Research* **248**(3), 815–826.

Yıldız, B. and Karaşan, O. E. (2017), 'Regenerator location problem in flexible optical networks', *Operations Research* **65**(3), 595–620.

Yıldız, B. and Savelsbergh, M. (2019), 'Provably high-quality solutions for the meal delivery routing problem', *Transportation Science* **53**(5), 1372–1388.