
This is the **accepted version** of the journal article:

Bayliss, Christopher; Juan, Ángel A.; Currie, Christine S.M.; [et al.]. «A learnheuristic approach for the team orienteering problem with aerial drone motion constraints». Applied Soft Computing Journal, Vol. 92 (July 2020), art. 106280. 19 pàg. DOI 10.1016/j.asoc.2020.106280

This version is available at <https://ddd.uab.cat/record/296841>

under the terms of the  license

A Learnheuristic Approach for the Team Orienteering Problem with Aerial Drone Motion Constraints

Christopher Bayliss¹, Angel A. Juan¹, Christine S.M. Currie², Javier Panadero¹

¹ IN3, Universitat Oberta de Catalunya & Euncet Business School, Spain

² Mathematical Sciences Dept., University of Southampton, UK

Abstract

This work proposes a learnheuristic approach (combination of heuristics with machine learning) to solve an aerial-drone team orienteering problem. The goal is to maximise the total reward collected from information gathering or surveillance observations of a set of known targets within a fixed amount of time. The aerial drone team orienteering problem has the complicating feature that the travel times between targets depend on a drone's flight path between previous targets. This path-dependence is caused by the aerial surveillance drones flying under the influence of air-resistance, gravity, and the laws of motion. Sharp turns slow drones down and the angle of ascent and air-resistance influence the acceleration a drone is capable of. The route dependence of inter-target travel times motivates the consideration of a learnheuristic approach, in which the prediction of travel times is outsourced to a machine learning algorithm. This work proposes an instance-based learning algorithm with interpolated predictions as the learning module. We show that a learnheuristic approach can lead to higher quality solutions in a shorter amount of time than those generated from an equivalent metaheuristic algorithm, an effect attributed to the search-diversity enhancing consequence of the online learning process.

Keywords: team orienteering problem, metaheuristics, machine learning, learnheuristics, aerial drones, route-dependent edge times.

1 Introduction

In this work we solve a team orienteering problem (TOP) for a fixed-size fleet of aerial drones or unmanned aerial vehicles (UAV). The objective is to route each drone such that the total reward collected from visiting each of their assigned targets is maximised. Each drone must arrive back at the end depot before a maximum time limit –otherwise the rewards collected by that drone are lost. Furthermore, each target can only be visited once by one single drone. We consider a realistic variant of the TOP in which the travel times between targets are subject to physical constraints, including: air-resistance, ascent angle, gravity, and velocity reduction caused by turning. These constraints give rise to path-dependent travel times, which are the solutions to a set of equations of motion. The accurate prediction of inter-target travel times requires a detailed numerical approximation, which limits the efficiency of traditional metaheuristic approaches. Additionally, in the aerial-drone TOP it is possible that partial rewards can be achieved from making observations of targets from the locations of adjacent targets. Accordingly, we take partial observations into account based on the visibility of a target from an adjacent one relative to a direct observation of that target. We apply the emerging concept of learnheuristics [1, 2] to simultaneously optimise the routes of drones and learn to predict travel times. A learnheuristic approach integrates metaheuristics –as the optimisation module– and machine learning (ML) to rapidly predict solution-dependent decision costs using data obtained from a time-limited availability of detailed solution evaluations via a *reality module*. In particular, the travel time between two targets depends on the initial velocity on that edge, which is based on: (i) the final velocity at the end of the previous edge; and (ii) the turn angle required by the drone to start travelling towards the next

target. This is because sharper turns cause a greater decrease in velocity –an effect known as *scrubbing* within the context of motor racing. The required turn angle depends directly upon the node sequence. Furthermore, given an initial velocity for an edge traversal, the final velocity depends upon the forces acting over the drone during that edge traversal including: gravity, air-resistance, and the drones’ available thrust force for overcoming these resistive forces. Additionally, edge traversal times depend upon the gradient and length of the edge, since these influence the required amount of work against the force of gravity and the total distance the drone has to travel.

It is important to distinguish between travel times –which are dynamic due to path dependence, such as those considered in this work– and travel times with endogenous uncertainty –which are not considered in this work. Here, we consider edge-traversal travel times that can be deterministically determined from the path followed by a drone. The exact sequence of targets allocated to a drone directly influences the sequence of forces that act upon it. This, in turn, directly influences the speed of the drone at each subsequent moment in its tour. As a result, edge-traversal time is strongly path dependent. Figure 1 provides a flowchart of the proposed learnheuristic. The algorithm consists of iteratively generating candidate solutions using an optimisation module. This module employs a ML algorithm to approximate route-dependent travel times between targets. The most promising candidate solutions are then tested in the ‘reality’ module in order to accurately approximate their true objective value and associated decision costs. The ML algorithm uses this data to rapidly approximate decision costs in the metaheuristic search method. In each iteration, the newly generated solution is compared to the best overall solution to see if a new best overall solution has been found. Following this, the learning module is given access to the true decision costs, which are used to improve the accuracy of the ML module predictions in subsequent iterations. In a learnheuristic algorithm, an iteration consists of: (i) one execution of a metaheuristic algorithm –such as the exploration of the local neighbourhood of an incumbent solution; (ii) the use of a ML algorithm to evaluate decision costs; and (iii) the testing of a strong candidate solution in the reality module. The process continues until a specified maximum number of iterations have been completed. Notice that the budget of reality-module evaluations is the main computational bottleneck in the proposed learnheuristic algorithm. The use of a ML prediction model enables to evaluate intensive solution algorithms –such as metaheuristics or simpler heuristics, thus acting as a fast ‘proxy’ or surrogate model. In addition, the simultaneous learning and optimisation process gives rise to an additional beneficial search-diversification effect: in the early stages, the predictions provided by the ML algorithm will be rough approximations –i.e., additional diversification mechanisms may not be required. Using the time-expensive reality module alone limits the complexity of the search algorithms that can be employed. The key requirement of a ML module is that it can learn and generate cost predictions quicker than the reality module, ideally by one or two orders of magnitude. Additionally, the ML module also needs to be capable of closely approximating the reality module.

The optimisation module relies on an event-based constructive algorithm [3]. In this case, such an algorithm is employed for routing drones, whose decisions are controlled by the set of weights given to each of multiple decision criteria referred to as *efficiency attributes* (Section 7.1). The tours of drones are optimised using a two-phase metaheuristic approach, consisting of a multi-start variable neighbourhood search (VNS) algorithm followed by a biased-randomised (BR) algorithm. BR algorithms have been successfully applied to enhance the performance of constructive procedures in different vehicle and arc routing problems [4, 5], scheduling problems [6, 7], and facility location problems [8]. The optimisation module for the first half of the learnheuristic iteration budget consists of single iterations of the local search algorithm. In these iterations, the decision parameters are optimised. The remainder of the iteration budget is spent applying BR techniques to the promising sets of decision parameters found in the local search stage. The optimisation routine makes sparing use of the reality module. On the other hand, the optimisation algorithm makes frequent use of the ML module.

In the learnheuristic implementation presented in this work, the ML module is a fast instance-based learning algorithm with interpolated predictions. Instance-based algorithms are one of the simplest ML algorithms [9], and are based on storing all previous data and returning a prediction based on the stored data-instance whose initial conditions match those of the scenario for which a prediction is required. The interpolation step is an

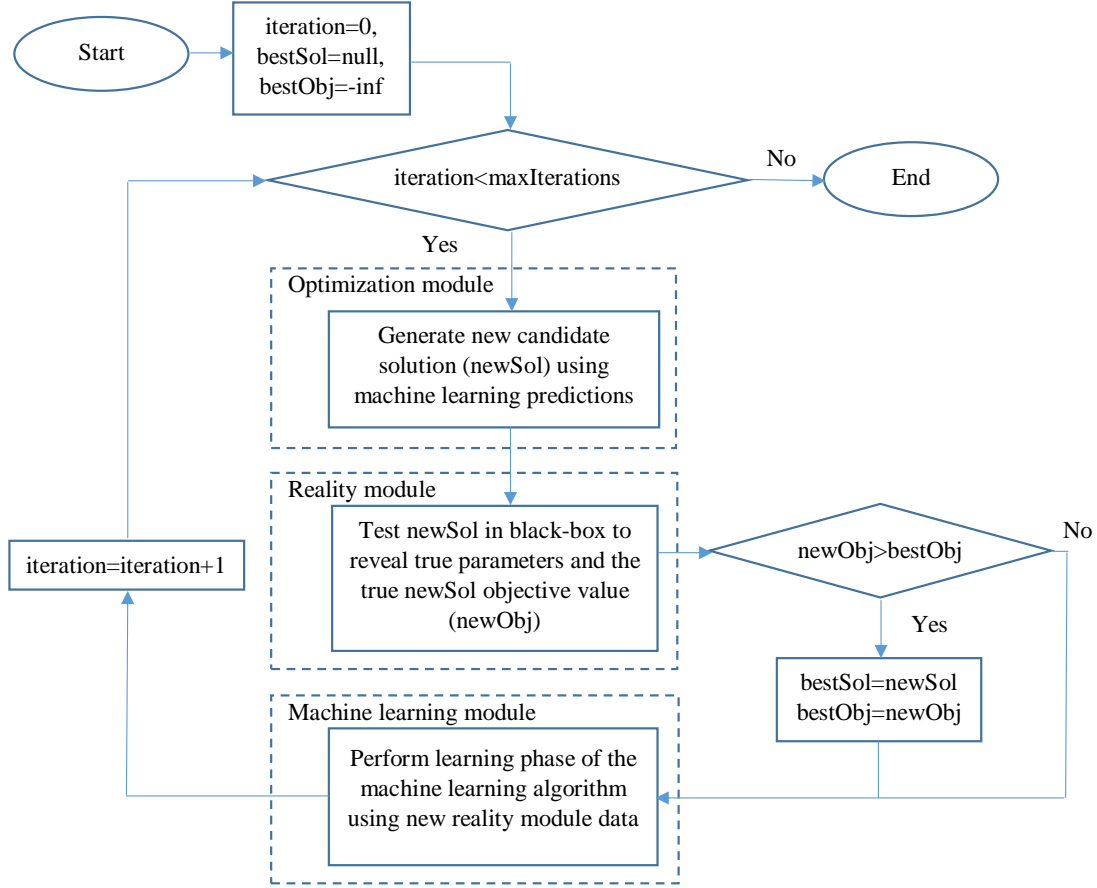


Figure 1: Overview of the proposed learnheuristic algorithm.

improvement over a standard instance-based method, as predictions are calculated from several of the nearest data points. Furthermore the interpolation method exploits numerous properties of the drone equations of motion to improve prediction accuracy (more details are provided in Section 6). To the best of our knowledge, the vast majority of existing research on team orienteering problems do not consider path-dependent edge-traversal times. Hence, they do not feature a learning problem in addition to an optimisation problem. As such, the main contribution of other non-learning algorithms lies in the quality of the optimisation module alone. The proposed learnheuristic approach, however, features an iterative framework based on: (i) ML-driven optimisation; (ii) reality-module testing of candidate solutions; and (iii) ML-based algorithm training. This cycle is repeated in each iteration of our learnheuristic algorithm. In comparison, non-learning based models have no need for such an elaborated mechanism, since they assume constant edge-traversal times regardless of the path followed by the drone. All in all, the main contributions of this article are as follows: (i) the introduction of an emerging and challenging real-world aerial drone route optimisation problem, which has path-dependent travel times; (ii) the development of a solving learnheuristic approach, which is able to outperform traditional metaheuristics – e.g., our learnheuristic can find solutions of equal or better quality in a fraction of the time; (iii) the use of an extended version of a sensitivity-based local search (optimisation module), which was initially introduced in [10]; (iv) the introduction of a problem-specific and instance-based learning approach, which uses interpolated data for predicting inter-node travel times – a model that exploits structural properties of the equations of motions to generate more precise predictions; and (v) a series of insights, such as experimental evidence that measures the benefits of integrating online learning and optimisation.

The remainder of the paper is structured as follows: Section 2 reviews related work on the TOP and the

topic of drone motion modelling. Section 3 formulates the team orienteering problem with aerial drone physical constraints. Section 4 discusses the use of ML methods as surrogate models and how these are related to the concept of learnheuristics. Section 5 provides details of the detailed numerical approximation scheme (reality module), while Section 6 introduces the ML module. Section 7 explains our event-based drone-routing algorithm. Section 8 describes the optimisation module of the learnheuristic algorithm. Section 9 contains the results of the computational experiments carried out to test our learnheuristic algorithm. Finally, Section 10 summarises the main findings and future research directions.

2 Related Work

This section provides a brief review of different topics directly related to our work, including: the team orienteering problem and some of its more popular variants, application of drones in logistics, as well as modelling vehicle motion in routing problems.

2.1 Single-Vehicle and Team Orienteering Problems

The single-vehicle *orienteering problem* (OP) was introduced by Golden *et al.* [11]. Being *NP-hard*, the majority of solving methods for the OP have relied on heuristics. Early work considered the simplest deterministic version of the OP, in which one vehicle chooses the set of nodes to visit –as well as the visiting order– during a specified time interval [12]. The OP has many applications, including the tourist trip design problem [13]. Some variants of this problem include the time-dependent OP [14]. Our work focuses on the TOP, an extended version of the OP in which a team of m vehicles aims to maximise their combined reward from visiting a selection of points within a given time limit. The problem was first introduced in [15], who extended their methodology from the single-vehicle OP to consider multiple vehicles. Some well-known variants of this problem are the TOP with time windows [16] and the multi-modal TOP with time windows [17]. Chao *et al.* [15] set up the TOP as a multilevel optimisation problem with three levels: select which points to visit, assign points to each member of the team, and determine the shortest path for each team member around the points they have been assigned. Such sequential approaches reduce the size of the overall problem at the risk of removing the optimal solution from the resultant solution space. Exact solutions have been obtained for mid-sized problems (up to 100 vertices) using an efficient column-generation algorithm [18]. Keshtkaran *et al.* [19] propose a branch-and-price algorithm and use dynamic programming to solve pricing problems. Similarly, El-Hajj *et al.* [20] apply a cutting planes approach, while Dang *et al.* [21] employ a branch-and-cut approach. Still, the vast majority of methods are based on heuristics. Hence, for instance, Campos *et al.* [22] use a greedy randomised adaptive search procedure (GRASP) with path relinking. Likewise, Ke *et al.* [23] provide a Pareto-mimic algorithm in which a Pareto front of non-dominated solutions is iteratively evolved. Their algorithm improved 10 best-known solutions to benchmark problem instances. Most of the recent approaches have been metaheuristics, including particle swarm optimization (PSO) [21], simulated annealing (SA) [24], harmony search [25] and evolutionary algorithms (EA) [26, 27, 28]. Lin [24] integrates an SA stage inside a multi-start procedure to enhance the diversity of the search. The algorithm begins with a randomly-generated initial solution before going into the iterative procedure. In each iteration, the algorithm selects a new solution from the neighbourhood of the current one. If the objective-function value of the new solution is better than that of the current one, the new solution replaces the current solution and the search process is resumed. As in any other SA structures, there is also a small probability that a new solution, with a worse objective-function value, may be accepted as the new current solution. Recent work on the OP and the TOP considers uncertainty in the rewards or the travel times [29, 30].

2.2 Drone Routing Applications

Amazon [31] and Google [32] have trial projects for drone delivery systems. Moormann *et al.* [33] considers a medication delivery application. Roberts *et al.* [34] consider the problem of trajectory optimisation for aerial drones used to generate 3D models of landscapes, where the drone’s trajectory greatly influences the quality of the data available for generating the model. Coutinho *et al.* [35] consider a disaster assessment problem in which they optimise the trajectories of gliders. Zhen *et al.* [36] consider an aerial monitoring application using UAVs, in which they optimise trip time and observation quality with monitoring height as a variable. Otto *et al.* [37] provide an extensive survey on the civil applications of unmanned aerial vehicles (UAV), highlighting –among other things– the emerging market potential of these vehicles and some new optimisation problems that their increasing use give rise to.

Another application area is in last-mile delivery. Murray and Chu [38] consider an application in which drones are deployed from trucks to cover “the last mile”. Here, mathematical programming models are provided for truck- and drone-route optimisation. Poikonen *et al.* [39] consider the vehicle routing version and provides extended results comparing: (i) total delivery times for traditional trucks-only formulations; and (ii) trucks with drones formulations. According to these authors, the main practical advantage of the drone vehicle routing problem is the ability to parallelise tasks (with a fleet of drones per truck) and the ability to take advantage of “as the crow flies” trajectories. Sundar and Rathinam [40] consider an UAV-based travelling salesman problem (TSP) with multiple recharge depots dispersed across the network. The objective being to visit all customers without running out of fuel and doing this in the shortest amount of time possible. They provide an approximation algorithm capable of achieving solutions that are within 1.4% of optimality for small to medium problem instances. Venkatachalam *et al.* [41] consider a drone fleet delivery problem under uncertain fuel usage in the presence of multiple refuelling depots. They propose a two-stage mixed integer programming model. The first stage determines the routes, the second stage considers realisations of the uncertain fuel usage and visits to refuelling depots as recourse actions. Dorling *et al.* [42] present exact and heuristic approaches to vehicle routing problems with aerial drones, their work empirically addresses the issue of the relationship between battery weight, payload, and energy consumption. Kim *et al.* [43] propose a robust optimisation algorithm for drone flight scheduling. They take battery life uncertainty into account and use a regression approach to model the influence of air temperature on battery life reduction.

2.3 Modelling the Motion of Vehicles in Routing Problems

Most existing approaches to modelling vehicle motion in routing problems are based on the work of Dubins [44]. This work deals with minimum path lengths under a curvature constraint, i.e., the maximum turn rate of a vehicle. In this version of the TOP, the route taken by a drone obeys the laws of motion. This means allowing for Newton’s laws of motion, gravitation, and the effects of air-resistance. Methods based on approximating the inter-target travel times by assuming constant drone velocities can lead to solutions that are impractical or infeasible. A consequence of the laws of motion is that a drone’s path is constrained: (i) the acceleration of the drone depends on the continual interaction of these forces; and (ii) a drone’s minimum turn radius depends on its speed. Our work focuses on (i) and approximates the effect of (ii). Regarding the former, the equations of motion are solved to determine the kinematics of drones on the paths between target locations. As regards as the latter, a velocity penalty –dependent on the turn angle– is applied. Regarding work on routing Dubins’ vehicles, Ny *et al.* [45] consider the curvature constrained travelling salesman problem and provide approximation ratios for tour lengths of Dubins’ vehicles in comparison to non-Dubins’ –or Euclidean distance tour length for the same TSP solution. Isaacs and Hespanha [46] provide a graph-based approach to the Dubins’ TSP. Savla *et al.* [47] provide worst-case tour length results for the Dubins’ TSP and also for a stochastic version, where targets are generated at random. Babel [48] proposed heuristic algorithms for the Dubins’ TSP with obstacles, in which a discrete routing model allows for full exploitation of an aircraft’s flight capabilities. In our work, we do not take the approach of modelling the motion of drones as Dubins’ vehicles. Instead, we assume that drones have the ability to slow

down to reduce their minimum turn radius such that all routes are feasible. This is a suitable assumption for aerial drones with the ability to hover, such as quadrotors.

3 Problem Formulation

This sections provides a mathematical formulation of the TOP for aerial surveillance drones. The problem is defined on a complete directed graph $G = (N, E)$, where N is a set of all nodes –including all potential target nodes as well as the start and end depots–, and E is the set of edges joining each pair of nodes. A set of drones B , all of which are initially stationed at a starting depot n_0 , are required to visit a subset of targets $n \in M = N \setminus \{n_0, n_{|N|}\}$, with location p_n and surveillance (reward) values $s(n)$ ($\forall n \in M$). The fleet of drones has a fixed amount of time ($T_{max} > 0$), in which to maximise the accumulated value derived from surveillance observations before returning to the end depot ($n_{|N|}$). *If a drone fails to reach the end depot, the surveillance rewards attained by that drone are lost.* A reward is collected from a visit to a target if it is visited directly once. After this, no additional value is available from that target. If a target is not visited directly, then a partial reward is possible. This partial reward depends upon the distance of that target from the nearest target that is visited directly by a drone. The partial reward that can be collected, from an observation of node k , when visiting a node j is based on the relative visibility of node k compared to a direct visit to that node. In particular, we assume that drones have a fixed cruising altitude H , and that the nodes have a width of W . Then if node k is a distance l_{jk} away from node j , then the partial reward q_{kj} achieved for a partial observation of node k from node j is closely approximated by elementary trigonometry as in Equation 1. We assume that only the closest observation of an unvisited node contributes to the total score objective.

$$q_{kj} = s(k) \left(\frac{\tan^{-1} \left(\frac{W}{\sqrt{l_{jk}^2 + H^2}} \right)}{\tan^{-1} \left(\frac{W}{H} \right)} \right) \quad (1)$$

For simplicity, we also assume that *drones are restricted to travelling directly between nodes*. This means that, upon reaching a target, the drone’s velocity vector changes instantaneously, so that the drone’s motion is directly towards the next target. However, the effect of turns on the magnitude of a drone’s velocity is taken into account using a velocity penalty, which increases with turn angle. This approach approximates the concept referred to as “scrubbing” in motor racing, and helps to avoid the need to solve the control problem of driving the drones –a problem that is explicitly considered by Zulu and John [49]. Our work considers the aerial drone TOP in which drone velocities, at any given point in a tour, depend on the prior path followed by the drone up to that moment. This property gives rise to path-dependent travel times. In particular, the path-dependent travel times of drones arise due to the effects of the continuous dynamic interaction of the drones’ thrust force with the forces of gravity and air-resistance. Section 5 provides full details of the equations of motion of aerial drones under the influence of these forces.

3.1 Mathematical Programming Formulation

The optimisation problem can be formulated as follows, where v_{bki} is a binary integer decision variable which takes a value of 1 if edge e_{ij} is the k^{th} edge traversed by drone b . The objective function, Equation (2), is to maximise the total reward collected by a fleet of drones B from direct and partial observations of a set of targets M . The first term of Equation (2) accounts for direct observations of targets, whereas the second term accounts for the best single-partial observation of targets not visited directly by a drone. The inclusion of partial observations makes the formulation represented in Equations (2) to (9) a non-linear mathematical program:

$$\max \sum_{b \in B} \sum_{k \in K} \sum_{i \in N} \sum_{j \in N} v_{bki j} \cdot s(j) + \sum_{j \in N} \left(1 - \sum_{b \in B} \sum_{k \in K} \sum_{i \in N} v_{bki j} \right) \max_{\substack{b' \in B \\ k' \in K \\ i' \in N}} (q_{ij} \cdot v_{b'k'i'j}). \quad (2)$$

The routes of drones are constrained as follows. Constraints (3) ensure that the first node in any route is the start depot (node 0) and that the end depot is entered from a non-depot node once in any route:

$$\sum_{j \in N \setminus \{0\}} v_{b00j} = \sum_{k \in K} \sum_{i \in N \setminus \{|N|\}} v_{bki|N|} = 1, \forall b \in B. \quad (3)$$

Constraints (4) ensure that each non-depot node is entered and exited an equal number of times and no more than once, which also ensures that only a single reward can be gained from any non-depot node:

$$\sum_{b \in B} \sum_{i \in N} \sum_{\substack{k \in K \\ i \neq j}} v_{bki j} = \sum_{b \in B} \sum_{i \in N} \sum_{\substack{k \in K \\ i \neq j}} v_{bkji} \leq 1, \forall j \in N \setminus \{0, |N|\}. \quad (4)$$

Constraints (5) ensure that only one edge is traversed in each position of each route:

$$\sum_{i \in N} \sum_{j \in N} v_{bki j} \leq 1, \forall b \in B, \forall k \in K. \quad (5)$$

Constraints (6) ensure that non-depot nodes are not visited more than once in total:

$$\sum_{b \in B} \sum_{k \in K} \sum_{i \in N \setminus \{0\}} v_{bki j} \leq 1, \forall j \in N \setminus \{|N|\}. \quad (6)$$

Constraints (7) are route continuity constraints, and ensure that entered non-depot nodes are immediately exited in the next edge traversal. Constraints (3)-(7) ensure that solutions consist of feasible tours:

$$\sum_{i \in N} v_{bki j} - \sum_{h \in N} v_{i(k+1)jh} = 0, \forall b \in B, \forall k \in \{1..|K| - 1\}, \forall j \in N \setminus \{0, |N|\}. \quad (7)$$

Constraints (8) ensure that each vehicles' route is completed within the time limit. Edge-traversal times depend on the route followed previously by a drone. This route is expressed by the following function $f(\cdot)$:

$$\sum_{i \in N} \sum_{j \in N} \sum_{k \in K} v_{bki j} f(v_{blmn}), \forall m \in N, n \in N, l \in K | l < k \leq T_{max}, \forall b \in B. \quad (8)$$

Constraints (9) express that the v variables are all binary variables:

$$v_{ijmn} \in \{0, 1\}, \forall i \in I, \forall j \in J, \forall m \in N, \forall n \in M. \quad (9)$$

Since the formulation (2)-(9) has a non-linear objective function and solution dependent decision costs (Constraint (8)), linear programming solvers are not an option. In the learnheuristic solution described in the next sections, the routes of drones are defined as the sequence of nodes assigned to them, where Y_{bk} is the k^{th} node in drone b 's tour. Y_{bk} variables are related to $v_{bki j}$ variables by $Y_{bk} = \sum_{i \in N} \sum_{j \in N} i v_{bki j}$.

3.2 Solution Evaluation

In this work, heuristically-generated candidate drone tours Y are evaluated according to the $TotalReward(Y)$ function provided in Algorithm 1.

This algorithm takes as input the set of drone tours to be evaluated, Y_b ($\forall b \in B$), and returns the total rewards collected by the drones. It maintains a list of non-visited nodes, U , which is initially set to those nodes not included in any drone route (line 3). It also maintains a list of drones, \mathcal{B} , which complete their assigned tour within the

Algorithm 1 *TotalReward*(Y)

```
1: Inputs: drone routes  $Y_b \forall b \in B$ 
2: Initialise total reward,  $totalReward = 0$ 
3: Initialise the set of unvisited nodes,  $U = N \setminus \bigcup_{b \in B} \{i \in Y_b\}$ 
4: Initialise the set of drones that successfully finished their tours within the time limit,  $\mathcal{B} \leftarrow \emptyset$ 
5: for  $b \in B$  do
6:   Evaluate drone tour time using Algorithm 2
7:    $\tau_b = TourTime(Y_b)$ 
8:   if  $\tau_b \leq T_{max}$  then
9:      $totalReward \leftarrow totalReward + \sum_{i \in Y_b} S(i)$ 
10:     $\mathcal{B} \leftarrow \mathcal{B} \cup \{b\}$ 
11:   else
12:     $U \leftarrow U \cup \{i \in Y_b\}$ 
13:   end if
14: end for
15: Add rewards for partial observations of nodes not visited directly by drones that finished their tours successfully
16: for  $u \in U$  do
17:    $totalReward \leftarrow totalReward + \max_{\substack{b \in \mathcal{B} \\ i \in Y_b}} q_{iu}$ 
18: end for
19: Output:  $totalReward$ 
```

time limit T_{max} . The algorithm then considers each drone route in turn, and evaluates the time required to complete each route using the $TourTime(Y_b)$ function, which will be described in the following. If the tour time τ_b of drone b is less than or equal to T_{max} , then the rewards for direct observations of the nodes in that tour are added to the $totalReward$ (line 9), and the drone is added to list \mathcal{B} (line 10). If τ_b exceeds T_{max} , then a route failure has occurred, and the nodes in that route are added to the list U (line 12). After evaluating the tours of each drone (line 5-14), the algorithm accounts for any rewards from the best partial observations of any non-visited nodes U by drones which successfully completed their routes within T_{max} . The time taken for a drone to traverse an edge is dependent upon the initial velocity, gradient, and length of the edge. Furthermore, the initial velocity on each edge depends upon the final velocity on the previous edge and the turn angle between the previous and the current edge. In this case, the time taken for a drone to complete its tour is calculated as in Algorithm 2, where: β_{ij} denotes the gradient –with respect to horizontal ground– of the edge from node i to node j , and l_{ij} is the length of the edge between nodes i and j .

Algorithm 2 simulates the sequence of edge traversals in a drone’s route by solving the drone equations of motion for each edge (lines 6 and 7), whilst applying turn penalties for the turns performed at each node visited between the start and end depot nodes (lines 10 and 11). The initial velocity u of a drone on an edge is calculated by applying the turn penalty to the final velocity v of the drone when it reached the end of the previous edge. Additionally, the initial velocity on an edge is limited to the terminal velocity on the current edge, which amounts to assuming that any necessary deceleration occurs instantaneously. The turn penalty values are described in detail in Section 5.1. The tour time (τ) is the sum of the times to traverse each edge. Section 5 provides the details of the drone equations of motion and their numerical solution.

4 Surrogate Models and Learnheuristics

The section discusses hybridisation in optimisation, existing examples of ML methods used as surrogate objective functions, and a discussion of the relationship between learnheuristics and reinforcement learning.

Algorithm 2 *TotalTime* (Y_b)

```
1: Inputs: drone path  $Y_b$ 
2: Initialise the drone's initial velocity  $u = 0$  and total tour time  $\tau = 0$ 
3: for  $j \in \{Y_b \setminus \{n_0\}\}$  do
4:    $i =$  previous node to  $j$  in path  $Y_b$ 
5:    $k =$  next node to  $j$  in path  $Y_b$ 
6:    $u \leftarrow \min(u, \text{terminalVelocity}(e_{kj}))$  (see Equation (25))
7:   // Solve the equations of motion to determine  $t$  the edge traversal time and  $v$  the final velocity on the current
   edge, denoted by the function  $h()$  (see single edge traversal simulation Algorithm 3)
8:    $(t, v) \leftarrow h(u, \beta_{ij}, l_{ij})$ 
9:   // Update total tour time
10:   $\tau \leftarrow \tau + t$ 
11:  // Apply turn penalty ( $\pi_{ijk}$ ) for traversing edge  $e_{jk}$  immediately after edge  $e_{ij}$  to determine the drones initial
   velocity on the next edge
12:   $u \leftarrow v\pi_{ijk}$ 
13: end for
14: Output:  $\tau$ 
```

4.1 Fitness Function Approximation

In many real-world engineering and combinatorial optimisation problems, complex simulation models are required for the evaluation of candidate solutions. The substantial time requirements of such detailed simulation models precludes the use of methodologies that require a high number of function evaluations such as metaheuristics. ML algorithms are often used to provide fast fitness function approximations. Learnheuristics [2] provide a formal framework for such an approach, with an emphasis on combinatorial optimisation problems with an inherent parameter learning problem. Many fitness function approximation methods have been proposed. Jin [50] provides a comprehensive survey and states that the most popular methods of fitness function approximation are: “the response-surface methodology, the kriging model, the feed-forward neural networks, [...] and support vector machines”. However, since they typically require extensive computing times, these ML methods do not always fit well in the learning module of a learnheuristic. For that reason, we propose an approach that uses a fast and easy-to-implement instance-based learning algorithm. It iteratively learns to approximate solution-dependent decision costs, from a limited budget of full fitness function (reality module) evaluations, during the optimisation process. Such an approach gives rise to the possibility of selecting solutions that will provide the learning algorithm with training data. During the iterative solution process, a diminishing weight is given to selecting edge traversals based on a prediction uncertainty measure of the estimated edge-traversal time (Section 7.1).

Other approaches to fitness-function approximation include [51], who use k -means clustering on the population of solutions and perform full fitness-function evaluations only for elements nearest to the cluster centroids—the other elements are approximated with an ensemble of neural networks. Their algorithm significantly outperforms the plain evolutionary strategy with the same number of full fitness-function evaluations and without any fitness-function approximations. A similar comparison is made in this work (Section 9). Brownlee *et al.* [52] propose the use of a Markov's network as a surrogate objective function in a genetic algorithm, and find that such an approach is only beneficial for cases where the fitness functions are substantially complex. In their work, the proposed fitness-function approximation is close to two orders of magnitude faster than the fully-detailed evaluation approach. Lim *et al.* [53] introduce a generalised framework for utilising a variety of surrogate models within an evolutionary algorithm. The weights given to each surrogate model are based on the accuracy of the fitness predictions generated in previous iterations of the evolutionary algorithm. Yang *et al.* [54] use support vector machine regression to model the original objective function and accelerate the metaheuristic computing time.

4.2 Learnheuristics and Reinforcement Learning

Our learnheuristic framework is analogous to reinforcement learning in several ways –although the former is specifically adapted to the area of metaheuristics. Reinforcement learning is concerned with the problem of simultaneously learning and optimising the behaviour of agents in an environment through trial and error. The standard reinforcement learning framework [55] consists of an agent in a feedback loop in which actions are selected based on a policy that is conditioned on the agents state in the environment. For each action performed, a state transition occurs, and the agent receives a reward –which is used to update the agents policy for that state. The learning challenge is that of finding an optimal balance between exploration of different actions and the accumulated reward. Reinforcement learning algorithms are typically based on the formalism of Markov decision processes [56], whereby the states of a system exhibit the Markov property. Given this property, the Bellman equations [57] characterise the value function for an optimal agent, which defines the expected total reward associated with each state at each time. The value function can be used by the agent to make step-by-step optimal decisions –e.g.: by selecting the action with the maximum value of the immediate reward plus the expected value of the resultant state. Learnheuristics do not directly build upon the same standard mathematical formalism. Instead, they provide a framework for hybridising existing metaheuristics and learning algorithms in order to address optimisation problems with inherent learning requirements. Just as in reinforcement learning, learnheuristics are faced with the trade-off between exploration and exploitation when simultaneously learning and optimising. With its roots in the adaptive experiment design [58], the study of the multi-armed bandit reinforcement learning problem [59] has contributed much to address this trade-off. In particular, the Gittins’ index rule [60] was proven to provide an efficient solution to this problem. It provides a measure of the value associated with selecting an action in terms of the trade-off between the immediate reward and the information that will be gained from exploring that action. Just as a Gittins’ index accounts for the amount of previous exploration of a candidate decision, in the proposed learnheuristic algorithm we include a prediction uncertainty measure as a decision criterion for selecting targets-to-visit-next within an event-based solution construction procedure (Section 7).

5 Reality Module

This section provides the equations of motion accounting for the effects of gravity, air-resistance, elevation change, and turn angles, along with a numerical approximation scheme for their solution. The basic forces governing flight are thrust, drag, lift, and gravity [61]. These forces can be modelled using Newtonian mechanics [62]. Force is a scientific measure of actions applied to a mass, which can change its velocity. A constant force changes a mass velocity at a constant rate, i.e., a constant acceleration or deceleration. Thrust provides the acceleration force, which allows a drone to overcome resistive forces of drag (or air-resistance) and gravity. Lift is the vertical acceleration force, which is required to overcome gravity and allow an aircraft to leave the ground. Lift is generated through aerodynamic effects between the air and wings (or propellers). An increased lift raises the air-resistance. We do not directly model aerodynamic effects because the problem of choosing the correct wing angle in order to achieve the desired resultant acceleration direction is an aspect of the drone control problem. In this work we assume that *drones have a constant thrust force F* . This assumption is shown in experiments to be an accurate approximation [42]. Drones are also subject to *air resistance, with a force that is proportional to v^2 (the drone’s velocity squared)*. The air resistive force is given by the equation αv^2 [63], where the air-resistance coefficient is given by $\alpha = \frac{1}{2}\rho C_D A$. In the previous expression, ρ is the density of air, A is the cross sectional area of the drone in its direction of travel, and C_D is the drag coefficient. *As stated in Section 3, we assume that drones travel along the shortest straight line paths between consecutive nodes, since this greatly simplifies the drone control problem.* The terrain over which the drone is taking observations may exhibit elevation changes. Therefore, the time to traverse edge e_{ij} also depends on the gradient of that edge β_{ij} due to the effect of the gravitational force (mg) on the drone’s mass m –where g is gravitational acceleration. Given these assumptions, a drone’s motion along an

edge e_{ij} is governed by the following equations:

$$\begin{aligned} r_x &= \alpha v^2 \cos(\beta_{ij}) - f_x, \\ r_z &= mg + \alpha v^2 \sin(\beta_{ij}) - f_z, \\ F^2 &= f_x^2 + f_z^2, \\ \beta_{ij} &= \tan^{-1}(r_z/r_x). \end{aligned} \tag{10}$$

Here, r_x and r_z are the components of the resultant force of the drone, the magnitude of which divided by the drone's mass m gives the acceleration of the drone along the edge. Also, r_z corresponds to the vertical lift force, while r_x corresponds to the propulsive force parallel to the ground. F is the magnitude of the force available to the drone, while f_x and f_z are its components. The third and fourth equations constrain the travelling direction of the drone and the total available thrust force, respectively. These equations also encapsulate our approach, thus avoiding the need to solve a complex aerial drone control problem. The first and second equations of the set of Equations (10) define the forces acting on the drone due to gravity and air resistance in the component directions. The third equation of the set of Equations (10) associates the total force available to the drone with the unknown components of that force. The fourth equation of the set of Equations (10) states that the direction of the resultant acceleration is constrained to the direction of the edge being traversed. The set of Equations (10) capture the main force components of a drone in flight. The unknown parameters are f_x , f_z , r_x , and r_z . Thus, there are four equations that need to be solved to find the aforementioned parameters. This can be achieved by substitutions and the solution to a quadratic equation –see Equation (11). The reality module solves these equations of motion using a numerical approximation based on calculating the resultant acceleration at small time intervals of δ . This process continues until the edge length has been traversed and the total traversal time and final velocity have been revealed. Algorithm 3 provides an outline of a single edge traversal approximation.

Algorithm 3 Numerical approximation of a single edge traversal: $(t, v) \leftarrow h(u, \beta, l)$

```

1: Inputs: drone initial velocity  $u$ , edge gradient  $\beta$ , edge length  $l$ 
2: Predefined constants: time step size  $\delta$ , gravitational acceleration  $g$ , coefficient of air-resistance  $\alpha$ 
3: Initialise drone's position on the edge  $position = 0$ , the drone's speed  $s = u$ , and edge traversal time  $t$ 
4: while  $position < l$  do
5:   // Evaluate force equations to determine the drones acceleration  $a$ 
6:    $a \leftarrow \phi(s, \beta, g, \alpha)$ 
7:   // Update the drones speed and position based on the calculated acceleration  $a$ 
8:    $position \leftarrow position + s\delta + 0.5a\delta^2$ 
9:    $s \leftarrow s + a\delta$ 
10:  // increment the edge traversal time  $t$ 
11:   $t \leftarrow t + \delta$ 
12: end while
13: Output:  $t$  and  $s$ 
```

Lines 8 and 9 provide the simplest numerical approximation, the Euler's method. In this work we use the fourth order Runge-Kutta method for second order differential equations [64], since it provides a more accurate approximation. Another important issue is the choice of the time step size (δ). In experiments of the steepest downhill edge, it was found that the fourth order Runge-Kutta method did not need a step size in excess of $\delta = 0.1$. The ϕ function is evaluated by solving equations (10), as described above, resulting in the following (broken down) equations.

$$\phi(s, \beta, g, \alpha) = \frac{\sqrt{(b - f_x)^2 + (a - f_z)^2}}{m}, \tag{11}$$

where, $f_z = \sqrt{F^2 - f_x^2}$, $f_x = \frac{-g + \sqrt{g^2 - 4fc}}{2f}$, $a = mg + \alpha s^2 \sin(\beta)$, $b = \alpha s^2 \cos(\beta)$, $c = \tan(\beta)$, $d = cb - a$, $e = d^2 - F^2$, $f = c^2 + b$ and $g = 2cd$.

5.1 Turn Angle Effect

To model the effect of turns on velocity, we use a velocity penalty function that increases with turn angle. Suppose that part of a drone's route includes visiting nodes i , j , and k in sequence. Let θ_{ijk} denote the angle between edges e_{ij} and e_{jk} , and equals $\cos^{-1} \left(\frac{((p_j - p_i)(p_k - p_j))}{|p_j - p_i||p_k - p_j|} \right)$, where p_i is the position vector of target i . The turn penalty π_{ijk} , upon reaching node j , is $\pi_{ijk} = \cos(\theta_{ijk}/2)$. So, if a 180 degree turn is required, all of the velocity is lost in the turn. Also, if there is no turn, there is no velocity penalty. The initial velocity u on edge e_{jk} can be calculated as $u = v\pi_{ijk}$, where v was the final velocity on the preceding edge e_{ij} .

6 Machine Learning Module

Using the reality module to evaluate decision costs might require time-consuming numerical approximations (e.g., complex simulations), which prohibit the use of evaluation-intensive metaheuristic algorithms. Since accurate evaluations of drone tour times are the dominant computation bottleneck, the role of ML in our learnheuristic approach is to provide rapid predictions of path-dependent decision costs. Within a learnheuristic, the ML module is part of a feedback loop that also involves the optimisation module. Within each iteration of the learnheuristic, the following steps occur: (i) the optimisation module proposes a promising candidate solution, which is generated based on decision-cost predictions generated by the ML algorithm; (ii) this candidate solution is then tested in the reality module to reveal its true cost, in addition to accurate data regarding the decision costs; and (iii) the accurate data regarding decision costs is then used to train the ML algorithm. Hence, if the candidate solution is generated based on inaccurate predictions, then the data collected from the reality module can be used to improve those predictions. Thus, if a candidate solution turns out to be not so promising, then improved predictions should prevent the optimisation module from generating the same solution in subsequent learnheuristic iterations. This means that the integration of iterative optimisation and ML algorithms configure a search-diversification mechanism.

The learning component of the proposed learnheuristic framework has the role of rapidly predicting edge-traversal times and final velocities on those edges. The ML module has access to data provided by a limited number of reality-module solution evaluations. We propose an instance-based learning algorithm for playing the role of the ML module. This means that edge traversal predictions will be based on the historical edge-traversal data that is closest to the initial conditions of the edge traversal being predicted. Furthermore, instead of merely returning a prediction based on the nearest existing data, as in a standard instance-based approach [9], the predictions are interpolated from the nearest available data points. The ML module is decomposed by edge, which means that a separate instance of the instance-based learning algorithm is implemented for each edge and each edge-traversal direction. In contrast to the alternative approach of implementing a single-data value of the instance-based learning algorithm for making all of the required predictions, this decomposition approach eliminates edge length and gradient as independent variables for each edge prediction model—as these are constant in the data collected for each edge. Whilst this approach increases the number of individual learning problems, it significantly simplifies the prediction problem to that of predicting edge-traversal times as a function of a single variable—that of the initial velocity on the edge. Furthermore, each individual model is smaller with lower data requirements. The ML module also predicts the final velocities of drones after traversing each edge, as these are required for estimating the initial velocities on subsequent edges after taking turn-penalties into account. The prediction problem for each edge e_{ij} can be represented as follows:

$$(t, v) \leftarrow h'_{ij}(u). \quad (12)$$

Here, t and v denote the predicted traversal time and final velocity, respectively. Also, h' represents the prediction method as a function—in contrast to h , the actual edge traversal function. Also, u denotes the drones'

initial velocity. Four time evaluations are performed by replacing the function $h(u, \beta_{ij}, l_{ij})$ on line 7 of Algorithm 2 with $h'_{ij}(u)$. Our ML module can be any ML methodology for regression, such as a neural network [65], or regression trees [66]. In this work we use an instance-based approach with interpolation due to: (i) the speed and simplicity of the learning and prediction tasks; and (ii) our interest in exploiting the characteristics of the specific learning problem to improve the accuracy of the solving approach, as described in Section 6.2.

6.1 Reality Module Data

In general terms, the ML module fits the function h to the edge-traversal data generated in the reality module edge-traversal simulations –such in a way that the average prediction error is minimised. When a candidate solution for the aerial drone TOP is evaluated in the reality module, data is collected regarding the initial velocity u , traversal time t , and final velocity v for each edge traversed. Note that, since we assume that drones travel on straight paths between consecutive nodes, velocity is modelled as a one dimensional vector. Therefore, speed equals velocity in the following. Let D^{ij} denote the complete set of data collected from all of the traversals of edge ij evaluated so far in the reality module. Let d_k^{ij} denote the k^{th} element of this set of data, which itself is a tuple containing the initial speed, traversal time, and final speed for that edge traversal. Such a tuple can be expressed as $(u(d_k^{ij}), v(d_k^{ij}), t(d_k^{ij}))$, where the functions $u(\cdot)$, $v(\cdot)$, and $t(\cdot)$ return the initial speed, final speed, and edge traversal time, respectively. Furthermore, the set D^{ij} is an ordered set in which the elements are sorted in ascending order of the initial speeds of each data tuple, where d_k^{ij} denotes the data tuple, corresponding to the traversal of edge ij , with the k^{th} lowest initial speed. This ordering facilitates the interpolation method, which is used to generate traversal time predictions, t' , and final speed predictions, v' , given an initial speed $|u'|$ on edge ij .

6.2 Interpolated Prediction Method

In a purely instance-based approach, given an initial speed $|u'|$, predictions for the traversal time t' and final speed $|v'|$ on an edge ij are generated by identifying the data tuple in the set D^{ij} with the nearest initial speed $u(d_{k'}^{ij})$ to $|u'|$. That is, we find the $d_{k'}^{ij}$ that minimises $u(d_{k'}^{ij}) - |u'|$. In order to produce more accurate predictions, traversal time and final speed predictions are interpolated. In particular, a blend of linear and gradient interpolations are used. A linear interpolation takes an initial-speed-difference-weighted-sum of the data tuples with the nearest initial speeds either side of $|u'|$. In the gradient based interpolation approach, gradients are estimated for the rate of change of traversal time $\left(\frac{\partial t}{\partial u}\right)$ and final speed $\left(\frac{\partial v}{\partial u}\right)$ with respect to changes in initial speed. These are used to generate interpolated predictions by extrapolating predictions from the data tuples with the nearest initial speeds either side of $|u'|$. Finally, a weighted sum of the linear and gradient-based interpolated predictions is used to generate the final prediction. Noting that the data for each edge is sorted in increasing order of initial speed, the interpolation method exploits the following properties of traversal times and edge final speeds as a function of initial speed:

$$t(d_k^{ij}) \geq t(d_{k+1}^{ij}), \quad (13)$$

$$v(d_k^{ij}) \leq v(d_{k+1}^{ij}). \quad (14)$$

Property (13) means that edge-traversal times, $t(\cdot)$, monotonically decrease with initial speed $u(\cdot)$. Property (14) means that edge final speeds, $v(\cdot)$, monotonically increase with initial speed. For an edge-traversal starting from an initial speed of $|u'|$, these properties imply the following: when predicting the traversal time t' and final speed $|v'|$, the data point d' , with $u(d') = |u'|$, provides bounds on the prediction values t' and $|v'|$. Specifically, the data point d with the minimum initial speed that is higher than (or equal to) $|u'|$ (denoted \bar{d}), provides an upper bound for the final speed prediction, i.e., $|v'| \leq v(\bar{d})$. It also provides a lower bound for the

traversal time prediction, i.e., $t' \geq t(\bar{d})$. Similarly, the data point d with the maximum initial speed that is lower than (or equal to) $|u'|$ (denoted \underline{d}), provides a lower bound for the final speed prediction, i.e., $|v'| \geq v(\underline{d})$, as well as a lower bound for the traversal time prediction, i.e., $t' \leq t(\underline{d})$. It is possible to express \bar{d} and \underline{d} as follows:

$$\begin{aligned}\bar{d} &= \arg \max_{d \in D} \{u(d) \mid u(d) \leq |u'|\}, \\ \underline{d} &= \arg \min_{d \in D} \{u(d) \mid u(d) \geq |u'|\}.\end{aligned}\quad (15)$$

The bounds on the predictions can then be expressed as:

$$\begin{aligned}t(\bar{d}) &\leq t' \leq t(\underline{d}), \\ v(\bar{d}) &\geq |v'| \geq v(\underline{d}).\end{aligned}\quad (16)$$

The monotonicity of both edge-traversal time and edge final speed can be seen in Figure 2, which also shows the convexity of the functions. Convexity follows on from the fact that drone acceleration is decreasing in speed due to air resistance.

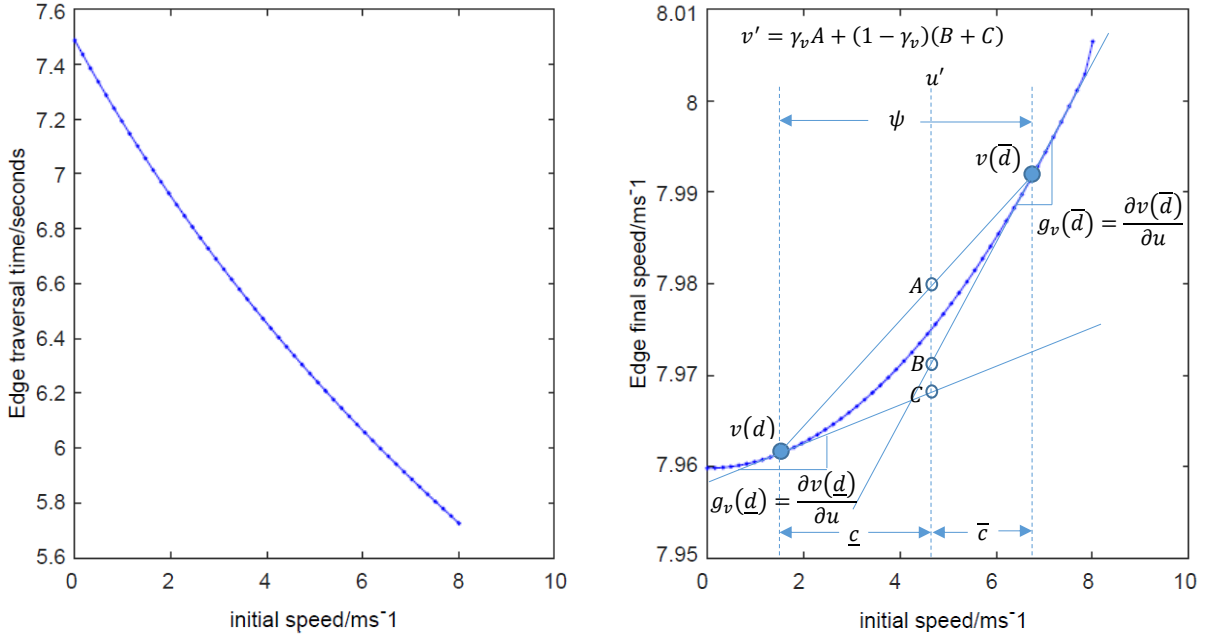


Figure 2: Monotonicity (and convexity) of edge traversal times and final speeds as a function of edge initial speed.

The convexity feature of both the traversal time and final speeds means that a linear interpolation is guaranteed to overestimate the true value. Letting $\psi = u(\bar{d}) - u(\underline{d})$, then $\bar{c} = |u'| - u(\bar{d})$, and $\underline{c} = u(\underline{d}) - |u'|$. In other words, the distance between initial speeds of the neighbouring data points and the initial speed for which a prediction is required. Notice that:

$$\left(\frac{\bar{c}}{\psi}\right)t(\underline{d}) + \left(\frac{\underline{c}}{\psi}\right)t(\bar{d}) \geq \hat{t}, \quad (17)$$

where \hat{t} denotes the true value for the edge-traversal time being predicted. The same applies to final speeds by interchanging $|v|$ with t . Now let $g_t(d_k)$ denote the gradient of the edge-traversal time function at the k^{th} lowest initial speed for a traversal of the given edge, which can be estimated from an initial speed distance weighted sum of forwards and backwards gradient approximations:

$$g_t(d_k) = \frac{\bar{c} \left(\frac{t(d_k) - t(d_{k-1})}{\underline{c}} \right) + \underline{c} \left(\frac{t(d_{k+1}) - t(d_k)}{\bar{c}} \right)}{\underline{c} + \bar{c}}. \quad (18)$$

The convexity of the traversal time and final speed functions means that gradient-based extrapolations starting from the neighbouring data points will underestimate the true value, i.e.:

$$\begin{aligned} t(\bar{d}) - \bar{c}g_t(\bar{d}) &\leq \hat{t}, \\ t(\underline{d}) + \underline{c}g_t(\underline{d}) &\leq \hat{t}. \end{aligned} \quad (19)$$

Therefore, a weighted sum of the linear interpolation and gradient-based extrapolations can result in a prediction of improved accuracy. Let γ_t and γ_v be the weight given to the linear interpolation for traversal-time and final-speed predictions, respectively (where $0 \leq \gamma_t \leq 1$ and $0 \leq \gamma_v \leq 1$). The final predicted edge traversal time (and final speed analogously) can be expressed by Equation (20).

$$t' = \gamma_t L_t + (1 - \gamma_t) G_t. \quad (20)$$

Here, L_t is the linearly interpolated traversal time prediction, given by Equation (21), and where G_t is the forwards and backwards extrapolation based interpolated traversal time prediction, given by Equation (22).

$$L_t = \frac{\bar{c}t(\underline{d}) + \underline{c}t(\bar{d})}{\psi}, \quad (21)$$

$$G_t = \frac{\bar{c}(t(\underline{d}) + \underline{c}g_t(\underline{d})) + \underline{c}(t(\bar{d}) - \bar{c}g_t(\bar{d}))}{\psi}. \quad (22)$$

Figure 3 shows that low values of the interpolation weight γ_t result in predicted edge-traversal times with the lowest root mean squared error (RMSE). Additionally, there is a weak negative correlation between RMSE and the solution quality –as measured by the total reward. The correlation coefficient was -0.1004 and the R^2 goodness-of-fit measure result was 0.0101 . Note that the predicted final speeds on edges are calculated with an analogous equation, where $t(\cdot)$ is replaced by $v(\cdot)$, and γ_t is replaced with a final speed interpolation weight σ .

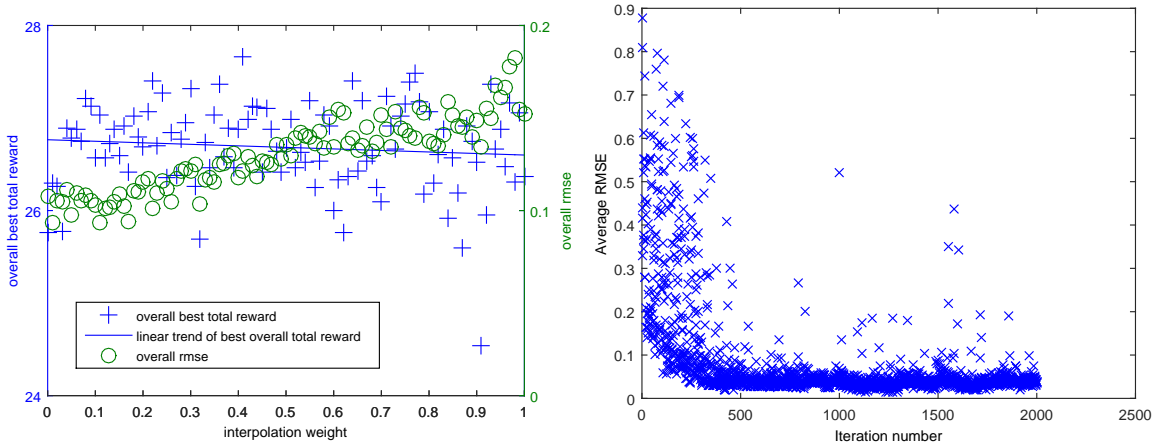


Figure 3: RMSE and reward by interpolation weight val-Figure 4: RMSE by iteration of the learnheuristic algorithm.

The edge-traversal time interpolation weight and the edge final-speed interpolation weight can both be learned from the prediction errors, which are revealed by testing a solution in the reality module. For this, exponential smoothing is used to learn the best value for an interpolation weight. For the case of the traversal-time interpolation weight, the exponential smoothing update is as follows:

$$\gamma_t \leftarrow \chi_t \gamma_t^* + (1 - \chi_t) \gamma_t, \quad (23)$$

where γ_t^* is the value of the interpolation weight that would have led to the correct predicted edge-traversal

time, and χ_t is the corresponding interpolation weight-learning rate:

$$\gamma_t^* = \frac{\hat{t} - G_t}{L_t - G_t}. \quad (24)$$

Here, \hat{t} is the true edge traversal time, G_t is the gradient-based interpolated traversal time –see Equation (22)–, and L_t is the linearly interpolated traversal time –see Equation (21). Note that, consistent with Figure 3, the traversal-time interpolation weight converges to a value close to 0.1, which means that gradient interpolations tend to be more accurate than linear interpolations, but still underestimate the true values slightly. Finally, Figure 4 illustrates how the RMSE of edge-traversal time predictions decreases as more iterations of the learnheuristic are performed. The ML module for each edge is initialised using the reality module to approximate with two edge traversals, one corresponding to an initial speed of 0 and another corresponding to an initial speed equal to the terminal velocity for that edge. This represents a moderate use of the reality module in comparison to the requirements of a metaheuristic algorithm based on reality-module-evaluations-only. The terminal velocity for an edge can be calculated analytically from Equations (10) by setting the resultant acceleration to zero ($r_x = 0$ and $r_z = 0$). It is given by the following equation:

$$terminal\ velocity = \sqrt{\left| \frac{-mg \sin(\beta) + \sqrt{m^2 g^2 (\sin^2(\beta) - 1) + F^2}}{\alpha} \right|}. \quad (25)$$

7 Event-Based Constructive Drone-Routing Heuristic

The general heuristic procedure that is used to build drones routes is summarised in Algorithm 4 and explained in more detail next.

Algorithm 4 is an event-based routing algorithm, similar to the one proposed by Fikar *et al.* [3] for the problem of home service routing with synchronised trip sharing. The event based approach is suitable in the current context due to the feature of solution dependence of inter-target travel times, which complicates the efficient use of typical local search moves such as those employed by the vast majority of the best metaheuristic algorithms for team orienteering problems without this feature [67]. In particular, we use local search moves based on the minimum changes to decision weights that modify a routing decision. Algorithm 4 generates routes for a fleet of drones simultaneously in time order. It maintains a list of drones sorted according to the earliest time drones will reach the most recent target allocated to them. In every iteration, the drone at the top of this list is assigned a next target to visit. The next target is selected by calculating efficiency scores for each unvisited target and selecting that with the highest score. The score for each unvisited node is computed as a weighted sum of efficiency attributes associated with the given drone visiting that node next. Section 7.1 explains the efficiency attributes in detail.

7.1 Efficiency Attributes of the Next Target to Visit

For each unvisited and feasible node, efficiency attributes (e.a.) are calculated on line 14 of Algorithm 4 in order to assess the relative benefits of selecting each candidate node as the next node a drone should visit. The e.a. are explained next:

1. **Reward (λ_1):** This e.a. is simply the reward that will be achieved from visiting the given candidate node. In problem instances where little time is available, greedily visiting high-value nodes can result in a very good solution.
2. **Traversal time (λ_2):** This e.a. takes on the value of the time required to reach the given candidate node, starting from the current node of the drone. Setting a high negative decision weight for this attribute results

in the selection of nodes that can be visited quickly. This may be due to a node being near, or due to the drone not having to slow down to make a sharp turn.

3. **Distance** (λ_3): This e.a. takes on the value of the distance, to the given candidate node, from the current node of the given drone. Setting a high weight for this attribute simulates a nearest neighbour algorithm.
4. **Reward per second** (λ_4): This e.a. is calculated as the reward associated with visiting a given node divided by the time required to reach that node, starting from the current node. It is a standard measure used in heuristic algorithms for the team orienteering problem.
5. **Velocity after turn** (λ_5): This e.a. takes on the value of the velocity after the turn that is required to travel towards a given candidate node. Setting a high weight for this efficiency attribute will lead to routes where drones fly along a straight path, avoiding speed reductions due to sharp turns.
6. **Velocity at next node** (λ_6): This e.a. takes on the value of the velocity upon reaching the given candidate node, starting from the current node. Setting a high decision weight for this attribute increases the velocity upon reaching the next node. This may be due to the next node being: at a lower altitude (reducing the required work against gravity), relatively far away (allowing the drone to accelerate for longer), or in the same direction the drone is already travelling (reducing the speed reduction due to a sharp turn).
7. **Edge-traversal-time prediction uncertainty** (λ_7): This e.a. is a measure of the level of uncertainty in the predicted time required to traverse the edge joining the drone's current location and the candidate target node. Setting a high weight for this attribute encourages the exploration of edges that have not been traversed regularly, which leads to more information being collected about them (via reality module testing). This, in turn, reduces their future prediction uncertainty level. As well as helping to improve the future accuracy of the ML module, this attribute also provides a search diversification mechanism. Edge-traversal time prediction uncertainty is calculated as follows:

$$\lambda_7 = (t(\underline{d}) - t(\bar{d})) \left(\frac{u(\bar{d}) + u(\underline{d}) - 2u'}{u(\bar{d}) - u(\underline{d})} \right). \quad (26)$$

The first term is an upper bound on the uncertainty of the traversal-time prediction, as the true traversal time must lie between the traversal times of the previous edge traversal instances (\underline{d} and \bar{d}), which had the nearest initial velocities below and above the current initial velocity on the same edge (u'). The second term is a measure of how close u' is to the midpoint between $u(\bar{d})$ and $u(\underline{d})$, where the prediction uncertainty is highest.

After calculating the e.a. scores for each feasible candidate node, all of these scores are normalised relative to the minimum and maximum scores for each e.a. Following this, the overall score for each candidate node is calculated as: $\sum_{i=1}^{|\lambda|} w_{bi} \lambda_i$ (see line 15 of Algorithm 4). The next section explains the algorithm used to search the space of e.a. weights using the event-based constructive algorithm presented in this section.

8 Optimisation Module

Within the learnheuristic framework set out in Figure 1, the optimisation module is tasked with determining the next solution that will be tested in the reality module. The main overall objective is to find a set of drone tours which maximise the total reward. However, there is the additional learning objective of ensuring that the ML module has a sufficiently diverse set of data. The latter objective is addressed by the use of an uncertainty measure for scoring candidate nodes (see λ_7 of Section 7.1). Equation (27) provides an exponential decay scheme for the weight given to prediction uncertainty in iteration f of the learnheuristic algorithm:

$$w_{b,7}(f) = w_{b,7}(0)\mu^f. \quad (27)$$

Here, μ takes on a value close to and less than 1. This approach means that the learning problem does not have to be treated as an additional objective, and the optimisation module can focus solely on maximising the main objective. In this paper, a two-phase metaheuristic algorithm is proposed. It consists of a multi-start local search algorithm –for searching the space of attribute weights– followed by a biased-randomised implementation of Algorithm 4. In this BR implementation, a strong candidate set of attribute weights –which were identified in the first phase– are explored in more detail. The local search component uses three local neighbourhoods, which are explained in Section 8.1. Another issue is the avoidance of multi-plicatively symmetrical solutions. For this, the weights are scaled proportionately after each modification, such that the absolute maximum weight value is 1.

8.1 Local Search Neighbourhoods

Equation (28) gives the minimum change to decision weight, w_{bj} , that modifies the overall highest-scoring candidate node from the current best candidate i^* to candidate node i . These minimum weight changes can be calculated for the current solution after line 24 of Algorithm 4. Here, ϵ is a positive value close to 0, and avoids a tie in the value of the candidate node i with the current best candidate node i^* .

$$\Delta w_{bj}^i = \frac{((score(i^*) - (w_{bj}\lambda_{i^*j})) - (score(i) - (w_{bj}\lambda_{ij})) + \epsilon)}{\lambda_{ij} - \lambda_{i^*j}} - w_{bj}. \quad (28)$$

The first local search neighbourhood consists of the absolute minimum change, to each of the decision weights individually –both positive and negative–, which change any of the routes generated by Algorithm 4 from those of the current solution. The minimum weight changes are made in Algorithm 4 (lines 23 and 24). For all $b \in B$ and for all $j \in \{1, 2, \dots, |\lambda|\}$, let n_1 denote the set of absolute minimum weight changes over all target choice decisions made in the current solution. The second local search neighbourhood consists of the average minimum weight changes for each of the decision weights individually, both positive and negative. In comparison to the first neighbourhood, this second neighbourhood represents a slightly larger step size. For all $b \in B$ and for all $j \in \{1, 2, \dots, |\lambda|\}$, let n_2 denote the set of average minimum weight changes over all target choice decisions made in the current solution. These first two neighbourhoods can be used as local search neighbourhoods for any problem in which a weighted-sum-of-efficiency-attributes approach is used, to make “next decisions”, within an event-based constructive algorithm. The third local search neighbourhood is different from the previous two, as it is based applying Algorithm 4 for subsets of drones sequentially. The initial implementation of Algorithm 4 is based on determining all drone routes simultaneously. However it is possible that generating routes for subsets of drones at a time may result in a higher-quality solution. If, for example, there are four drones, then the possible allocation orders can be represented as follows:

$$\begin{aligned} &(1, 1, 1, 1) \\ &(1, 1, 1, 2) \\ &(1, 1, 2, 2) \\ &(1, 2, 2, 2) \\ &(1, 1, 2, 3) \\ &(1, 2, 2, 3) \\ &(1, 2, 3, 3) \\ &(1, 2, 3, 4) \end{aligned} \quad (29)$$

The first allocation order given in Equation (29) corresponds to the default implementation of Algorithm 4. The second allocation order determines routes for the first three drones together, and then determines a route for the fourth drone involving the remaining unvisited targets last. The last one represents a fully sequential

implementation of Algorithm 4. The other allocation orders have analogous meanings. This neighbourhood is called the allocation order neighbourhood, and is denoted n_3 . In general, an allocation order neighbourhood can be used whenever a constructive procedure can be used to sequentially allocate tasks to a collection of agents, such as: aerial drones, vehicles, people, etc. Finally, let the set of local search neighbourhoods be denoted by \mathfrak{N} , i.e.: $\mathfrak{N} = \{n_1, n_2, n_3\}$. Defining a set of local search neighbourhoods is useful notationally for the following Algorithm 5, since local search neighbourhoods are temporarily ruled-out of consideration if they are known to not contain any solutions with an objective value higher than that of the incumbent solution.

8.2 Multi-Start Local Search Followed by Biased Randomisation

The following pseudo-code corresponds to the optimisation module of Figure 1. It is a multi-start local search based on the three local search neighbourhoods defined in Section 8.1. In each iteration, a local search neighbourhood is selected at random according to a probability distribution $P^{local\ search}$.

In an iteration of Algorithm 5, one of the currently unexplored neighbourhoods is selected and explored using the ML module. The best solution found in an iteration is then tested in the reality module. If the objective value of that solution $newObj$ is found to be the best solution found since the local search restarted, the new solution is set as the current solution and the set of unexplored neighbourhoods is reset. Additionally, the weights and allocation order for the new current solution are added to a set of strong candidates Z . It is these that are explored in more detail in the second phase of the metaheuristic algorithm via biased-randomisation. If $newObj$ is not an improvement upon the best solution since local search restarted, the current neighbourhood is removed from the set of unexplored neighbourhoods. If no unexplored neighbourhoods remain, it is concluded that a local optimum has been found and the local search restarts. The restart procedure can be either random or, alternatively, can be a weighted sum of a pair of strong candidate solutions in Z —the pair of strong candidates are selected using biased-randomisation, too. The output of a single iteration of the optimisation module is a new solution $newSol$ for the reality module solution to test (Figure 1).

The multi-start local search algorithm is used as the optimisation algorithm for half of the available iteration budget of the learnheuristic algorithm. In the second half, the set of strong candidate solution generation parameters Z are explored in more depth by applying biased randomisation on line 23 of Algorithm 4. As explained with examples in [68] and [69], this means that, instead of always selecting the candidate node with the highest total weighted sum efficiency attribute score, a node is selected using a skewed probability distribution such that higher scoring nodes have a higher probability of selection. In this work a *geometric probability* distribution is used to select the score rank index ($randI$) of the node in the score-ordered candidate list C that will be added to a drone's tour next. Equation (30) provides random candidate index for the *geometric* distribution, where ω is a parameter between 0 and 1. Notice that $\omega = 1$ corresponds to a full greedy behaviour, whilst lower values respectively lead to less greedy and more exploratory behaviour.

$$randI = Mod \left(\left\lfloor \frac{\log(\text{uniRand}(0, 1))}{\log(1 - \omega)} \right\rfloor, |C| \right). \quad (30)$$

Figure 5 shows that the best choice for the ω parameter of the biased randomisation implementation of the event-based routing algorithm is around 0.75, which corresponds to relatively greedy behaviour when choosing between candidate nodes to allocate to drone tours. Additionally, experiments revealed that focusing the biased randomisation on the top 10 candidate sets of solution generation parameters, with 100 repeated applications of the biased-randomised procedure, lead to the highest quality solutions. Figure 6 shows the current solution and the best overall solution in each iteration of an example implementation of the two-phase metaheuristic algorithm presented in this section. The overall average percentage difference between the best and current solution was 20.07%, and the standard deviation of the percentage difference between the best and current solution was 14.03%. For the first stage, these statistics were 25.08% and 17.13%, respectively. For the second stage, they were 15.06%

and 7.08%, respectively. These statistics reflect a first stage with a high level of search diversification, and a second stage with a higher level of search intensification.

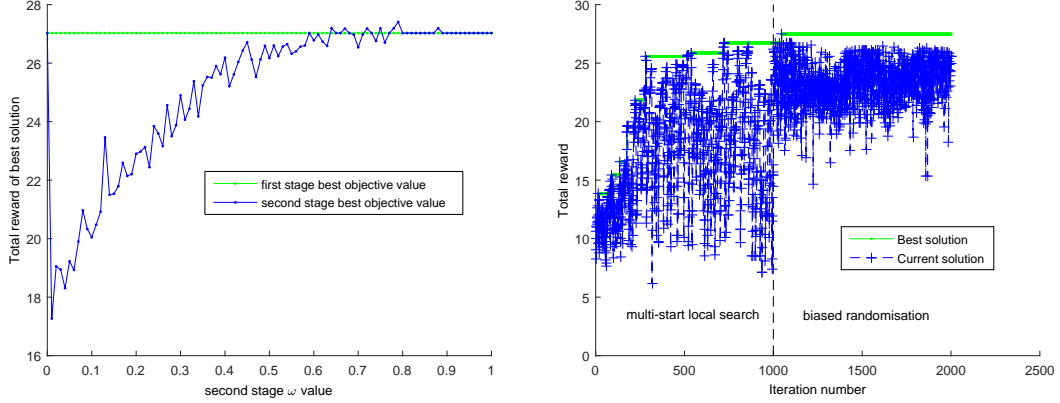


Figure 5: The effect of different β values on the best second stage total reward. Figure 6: Current versus best objective value by iteration.

9 Numerical Experiments and Results

In this section, we provide the results of numerical experiments which: (i) validate the proposed event-based drone-routing algorithm in the case of ‘static’ (not path-dependent) TOP instances; and (ii) compare the proposed learnheuristic algorithm with a reality-module-only metaheuristic –i.e., a reality-module only counterpart of the proposed learnheuristic and an *a priori* learning implementation of the proposed learnheuristic over a range of test instances, in which the level of path-dependency is varied. In the spirit of a fair testing, each algorithm has the same budget of reality-module simulation runs for testing candidate solutions. However the reality-module-only learnheuristic and the reality-module-only metaheuristic use additional reality-module runs as they use the reality module to generate candidate solutions using Algorithm 4, whereas the learnheuristic and the *a priori* learning implementation of the proposed learnheuristic use the ML module for this purpose.

9.1 Comparing the Learnheuristic with the Best-Known Solutions of the Static TOP

Tables 1 and 2 compare the proposed learnheuristic algorithm with the best-known solutions for the TOP benchmark standard instances. We present the results for the subset of benchmark instances considered by El-Hajj *et al.* [20], who provide optimal solutions for all but 5 instances, where relatively tight upper and lower bounds are provided. Since there are no existing benchmark TOP with path-dependent decision costs, this experiment validates the optimisation module in TOP benchmark standard instances.

All of the experiments reported in this work were executed on a desktop computer with a Core i5 - 8400 2.80GHz CPU with 8GB of RAM. The final three columns of Tables 1 and 2 refer to the solutions generated by our learnheuristic algorithm. In order to generate our solutions, a sample of metaheuristic search parameters was tested in each instance. We report the average and best solutions found by our algorithm over those samples. The samples of parameters explored different neighbourhood selection probability distributions, as well as the probabilities of random restarts and biased random restarts of the multi-start local search algorithm phase of the optimisation module. In all cases, 1,000 iterations of the local search phase followed by 1,000 iterations of the biased-randomisation phase were performed. The results show that the proposed learnheuristic is able to find good quality solutions in a matter of seconds for each of the benchmark instances. Furthermore, the proposed algorithm was able to find 23 optimal solutions out of 79 benchmark instances, and the overall average optimality

gap was 2.32%. Tables 1 and 2 show that the proposed solution approach attains relatively large gaps for the $p4$ instance set. These are relatively large instances and our results may benefit from an increased iteration limit. The results indicate that an event-based constructive routing algorithm, controlled by efficiency attributes weights with a second biased-randomisation phase, is a reliable method for generating high quality solutions rapidly.

9.2 Comparing the Learnheuristic with Reality-module-only Alternative Algorithms

The proposed learnheuristic algorithm is tested in a variety of physical settings, which have the effect of varying the degree of path-dependency of the edge traversal times of the test instances considered. For each physical setting, the proposed learnheuristic is compared with a reality-module-only metaheuristic algorithm, a reality-module-only version of the proposed learnheuristic, and an *a priori* learning version of the proposed learnheuristic.

- *Reality-module-only simulated annealing algorithm plus biased randomisation (RMSA)*: This is a simulated annealing algorithm [70, 71] based on the same set of neighbourhoods as presented in Section 8.1. It also uses a reality-module-only implementation of the BR phase, which contributes to enhance the SA approach in a similar way as described by Ferone *et al.* [72] for the GRASP. The reason for selecting a simulated annealing algorithm as a method of comparison for the proposed learnheuristic is to assess the benefits of the proposed learnheuristic approach compared to a standard metaheuristic approach for problems with solution-dependent parameters. Also, simulated annealing has been successfully applied to the team orienteering problem previously [24]. The algorithm required the setting of a temperature scheme. Experiments were performed to identify the best temperature scheme to use. Linear and exponential decay schemes and various initial temperatures were compared. It was found that an exponential decay based on an initial temperature of 10 and final temperature of 0.001 provided the best solutions. An initial temperature of 10 is reasonable, given that a useful rule for setting an initial temperature is that it should be close to the largest possible decrease in the objective value due to moving to a neighbouring solution in the local neighbourhood. The settings and results of this comparison are explained later in this section.
- *Reality-module-only learnheuristic (RMLH)*: This algorithm is identical to the proposed learnheuristic one except that the ML module is replaced with the reality module. The purpose of including such an approach in these experiments is to provide a reference point for judging the quality of the solutions generated by the proposed learnheuristic algorithm.
- *A priori learning heuristic (PH)*: This heuristic is identical to the proposed learnheuristic except that the learning phase is performed in an initial phase, where traversals of each edge are simulated starting from initial velocities between 0 and the terminal velocity for that edge as described in Equation (25). To ensure a fair comparison, the budget of edge simulations is the same as the total number of edge simulation performed in the corresponding LH experiment. Following this, the learnheuristic is implemented without considering additional edge-traversal data to the ML module. The purpose of including this method of comparison is to provide a benchmark for the learning aspect of the learnheuristic. In particular, it can be used to assess the impact on the solution quality of simultaneous learning and optimisation, such as that used in the proposed learnheuristic. The budget of reality module evaluations available in this approach is the same as that available to the learnheuristic (LH) approach.

LH, *RMLH*, *PH*, and *RMSA* are each implemented in a range of problem settings. The problem settings vary according to: (i) low and high air-resistance (drone cross-sectional areas $A = 0.04$ and 0.2 , respectively); (ii) zero and earth gravitational acceleration ($g = 0$ and 9.81 , respectively); (iii) turn-penalties off and turn-penalties on –when turn-penalties are off, drones do not have to slow down to turn; and (iv) partial observations off and partial observations on –when partial observations are off, only direct observations of targets result in a collected reward. Moreover, these problem settings are repeated for a 100 node example problem and a 150 node problem. Full details

of the test instances are available at www.researchgate.net/publication/338503289_LH_TOP_instances. The results in Table 3 are approximately ordered in increasing degree of path-dependent edge traversals –or, equivalently, in increasing order of physical interaction with the environment. This means that air-resistance had the largest effect on drones, followed by gravity, turn penalties, and partial observations. The results in Table 3 correspond to a problem instance in which the targets are positioned uniformly randomly within a 100 by 100 by 20 metre box. The rewards were selected from a uniform-random probability distribution. For each problem setting, there are four drones, each with a mass of $1kg$ and a thrust force of $1.05 \times 9.81N$ (the drone thrust force needs to exceed the gravitational force). The time limit T is set to 25 seconds. Table 3 provides the total rewards, solution times, and RMSE values for each of the four algorithms being compared in each problem setting –i.e., *LH*, *RMLH*, *RMSA*, and *PH*. The total rewards are calculated using the reality module. We provide solution-time results because these are relevant when considering that learnheuristics are designed for problems where time restrictions preclude intensive use of a time-expensive reality module. Since the role of the reality module can also be played by real-life experiments, the *LH* approach, with its reduced reliance on the reality module, can facilitate fast real-life optimisation of drone tours. Furthermore, fast solution time would indeed be beneficial if the set of nodes changes frequently. Additionally, lower solution times allow for additional ML and metaheuristic hyper-parameter tuning, which enables the identification of higher-quality solutions. We include RMSE error results for the *LH* and *PH* methods. The total reward results indicate that the proposed *LH* was able to find the best-known solution of the four algorithms being compared in 10 out of 16 test instances, and that in the remaining cases reasonably competitive solutions were identified. As shown in Figure 7, the four proposed algorithms provide similar results in terms of total rewards. However, the *LH* approach is able to reach high-quality solutions in a reduced amount of computing time. The RMSE of edge-traversal time predictions were calculated for the *LH* and *PH* methods.

The results for the 150 node problem are given in Table 4. These are based on the same drone configurations. The results in Table 4 follow a similar pattern to those in Table 3, except that the performance of the proposed *LH* demonstrates similar relative performance to the three algorithms of comparison –by achieving the highest average total reward with the best solution in 7 out 16 cases.

Algorithm 4 Event-based constructive drone-routing algorithm

```
1: Inputs: Efficient attribute weight matrix  $w$ ,
2: Reset drone routes  $Y_b \leftarrow n_0, \forall b \in B$ 
3: Reset the set of non-visited nodes  $U \leftarrow N \setminus \{depot, end\ depot\}$ 
4: Set  $nextArrivalTime(b) \leftarrow 0, \forall b \in B$ 
5: Let  $\mathfrak{B}$  be a list of drones ( $b \in B$ ) sorted in increasing  $nextArrivalTime$  order
6: while  $|\mathfrak{B}| > 0$  do
7:   // Select the drone ( $b$ ) expected to reach its last allocated node soonest as the drone that will be allocated an
   unvisited node next
8:    $b \leftarrow \mathfrak{B}_1$ 
9:   Reset the list of candidate nodes ( $C \leftarrow \emptyset$ ) that drone  $b$  can visit next
10:  // Consider the remaining unvisited nodes as potential candidates for the next node for drone  $b$  to visit next
11:  for  $n \in U$  do
12:    // Check the feasibility of drone  $b$  visiting node  $n$  using ML module edge traversal time predictions
13:    if drone  $b$  can visit node  $n$  and return to the end depot before time  $T_{max}$  then
14:      // Calculate efficiency attributes (Section 7.1) for drone  $b$  visiting node  $n$  next and the associated overall
      attractiveness score ( $score(n)$ )
15:       $score(n) \leftarrow \sum_{i=1}^{|\lambda|} w_{bi} \lambda_i$ 
16:      // Add node  $n$  to the candidate list  $C$ 
17:       $C \leftarrow C \cup \{n\}$ 
18:    end if
19:  end for
20:  if  $|C| > 0$  then
21:    // Sort the list of candidate nodes in order of decreasing attractiveness score
22:     $Sort(C, score, 'decreasing')$ 
23:    // Select the node with the highest attractiveness score ( $C_1$ ) as the node that drone  $b$  will visit next
24:     $Y_b \leftarrow (Y_b, C_1)$ 
25:    // Update drone  $b$ 's next arrival time according to the ML module's predicted edge traversal time to node
     $C_1$ 
26:     $nextArrivalTime(b) \leftarrow nextArrivalTime(b) + t' \left( Y_{b(|Y_b|-1)}, C_1 \right)$ 
27:    // Sort  $\mathfrak{B}$  in increasing order of  $nextArrivalTime$ 
28:     $Sort(\mathfrak{B}, nextArrivalTime, 'ascending')$ 
29:  else
30:    // No feasible candidate nodes exist for drone  $b$ , set the end depot as the drones next node and remove
    drone  $b$  from the drone list  $\mathfrak{B}$ 
31:     $Y_b \leftarrow (Y_b, n_{|N|})$ 
32:     $\mathfrak{B} \leftarrow \mathfrak{B} \setminus \{b\}$ 
33:  end if
34: end while
35: Output: drone routes  $Y$ 
```

Algorithm 5 One iteration of multi-start local search algorithm

```
1: Inputs: current solution ( $w$  (weights),  $a$  (allocation order)),  $bestObjValThisRestart$ 
2: Set  $newWBObj \leftarrow \inf$ ,  $newSol \leftarrow null$ ,  $newW \leftarrow \emptyset$ ,  $newA \leftarrow a$ 
3: Select a search neighbourhood  $n_k$  from  $\mathfrak{N}$  according to the probability distribution  $P^{local\ search}$ 
4: for  $m \in n_k$  do
5:   Set the candidate solution  $w', a'$  to neighbour  $m$  of the incumbent solution parameters  $w, a$ 
6:   Use Algorithm 4 to generate the drone routes  $Y'$  for the solution  $w', a'$  and obtain the (ML module based)
   objective value  $objVal'$  of that solution
7:   if  $objVal' > newWBObj$  then
8:      $newWBObj \leftarrow objVal'$ 
9:      $newSol \leftarrow Y'$ 
10:     $newW \leftarrow w'$ 
11:     $newA \leftarrow a'$ 
12:   end if
13: end for
14: Test  $newSol$  in the reality module to reveal its true objective value  $newObj$  (“reality module” of Figure 1)
15: if  $newObj > bestObjValThisRestart$  then
16:    $bestObjValThisRestart \leftarrow newObj$ 
17:   add  $(newSol, newW, newA)$  to  $Z$  (the decreasing objective value ordered list of strong candidate solutions
   and solution generation parameters)
18:   Reset  $\mathfrak{N} \leftarrow \{n_1, n_2, n_3\}$ 
19: else
20:   //Remove the explored neighbourhood  $n_k$  from the set of unexplored neighbourhoods  $\mathfrak{N}$ 
21:    $\mathfrak{N} \leftarrow \mathfrak{N} \setminus \{n_k\}$ 
22:   if  $|\mathfrak{N}| = 0$  then
23:     Restart the local search by selecting a random initial set of solution generation parameters (with proba-
     bility  $ProbabilityOfRandomRestart$ ) or by selecting a biased random pair of previous strong candidate
     solutions ( $Z$ ) and using their average as the initial solution.
24:     Reset  $\mathfrak{N} \leftarrow \{n_1, n_2, n_3\}$ 
25:     Set  $bestObjValThisRestart, newSol$  according to the new initial solution
26:     Evaluate  $newSol$  in the reality module to reveal  $newObj$ 
27:   end if
28: end if
29: Output:  $newSol, newObj$  (as required by the second decision block of Figure 1)
```

Table 1: Comparison of the event-based weighted sum routing algorithm with best-known solutions provided in [20] to the standard benchmark instance for the team orienteering problem.

Instance	nodes	Benchmark				Our solution					
		best LB	best UB	Gap	Time	best	Gap	Time	Average solution	Standard deviation	Total time
p1.2.p	30	250	250	0	7	240	4	2	238.9	2.08	401.1
p1.2.q	30	265	265	0	5	260	1.89	2	257.1	2.48	471.2
p1.2.r	30	280	280	0	4	275	1.79	2	268.4	2.70	484.2
Set Avg		265.0	265.0	0	5.3	258.3	2.56	1.9	254.8	2.42	452.2
p3.2.l	31	590	590	0	28	580	1.69	2	572.4	4.31	403.6
p3.2.m	31	620	620	0	33	620	0	2	604.0	5.42	487.4
p3.2.n	31	660	660	0	28	650	1.52	2	649.2	2.95	512.8
p3.2.o	31	690	690	0	19	690	0	2	687.9	4.06	496.7
p3.2.p	31	720	720	0	24	710	1.39	2	709.4	2.37	516.9
p3.2.q	31	760	760	0	12	750	1.32	2	735.0	6.26	554.6
p3.2.r	31	790	790	0	8	780	1.27	2	757.7	9.32	515.1
p3.2.s	31	800	800	0	0	800	0	2	789.7	4.67	568.4
p3.3.s	31	720	720	0	90	720	0	3	710.0	0.61	765.6
p3.3.t	31	760	760	0	42	740	2.63	3	721.9	7.50	802.0
Set Avg		711.0	711.0	0	28.4	704.0	0.98	2.2	693.7	4.75	562.3
p4.2.f	98	687	687	0	6550	647	5.82	25	629.8	7.28	6979.7
p4.2.h	98	835	835	0	2784	769	7.90	28	734.0	8.91	9579.5
p4.2.i	98	918	918	0	1064	864	5.88	39	833.9	9.01	10804.0
p4.2.j	98	965	965	0	2777	937	2.90	39	900.9	11.40	10727.6
p4.2.k	98	1022	1022	0	2751	975	4.60	51	947.4	9.26	12864.0
p4.2.l	98	1074	1074	0	7172	1019	5.12	60	995.4	6.81	13372.7
p4.2.m	98	1132	1132	0	4610	1071	5.39	56	1028.6	9.65	13192.9
p4.2.r	98	1292	1292	0	5016	1230	4.80	54	1209.2	11.40	16794.4
p4.2.t	98	1306	1306	0	0	1281	1.91	44	1263.3	6.01	17946.2
p4.3.g	81	653	653	0	52	615	5.82	28	596.7	6.52	7939.6
p4.3.h	90	736	736	0	801	695	5.57	32	678.2	7.40	10180.3
p4.3.i	94	809	809	0	2989	757	6.43	32	730.5	9.65	11347.0
p4.4.i	68	657	657	0	23	632	3.81	34	583.4	15.37	9619.4
p4.4.j	76	732	732	0	141	701	4.23	32	647.3	14.99	11891.6
p4.4.k	83	821	821	0	558	751	8.53	40	714.0	12.12	13614.3
Set Avg		909.3	909.3	0	2485.9	862.9	5.25	39.5	832.8	9.72	11790.2
p5.2.l	64	800	800	0	3	800	0	8	786.8	13.81	2174.0
p5.2.m	64	860	860	0	32	860	0	6	851.5	2.36	2350.5
p5.2.n	64	925	925	0	89	925	0	9	920.1	0.85	2663.1
p5.2.o	64	1020	1020	0	271	1020	0	14	1011.2	3.28	2936.5
p5.2.p	64	1150	1150	0	77	1150	0	14	1150.0	0	3011.5
p5.2.q	64	1195	1195	0	6597	1190	0.42	11	1190.0	0	3181.8
p5.2.r	64	1260	1260	0	123	1255	0.40	14	1250.0	0.75	3628.1
p5.2.s	64	1340	1340	0	845	1340	0	11	1322.1	4.18	3588.8
p5.2.t	64	1400	1400	0	418	1375	1.79	15	1363.8	4.41	3791.6
p5.2.u	64	1460	1460	0	3263	1450	0.68	19	1440.3	5.50	3929.1
p5.2.v	64	1505	1505	0	3497	1500	0.33	14	1488.1	4.56	3960.2
p5.2.w	64	1565	1565	0	5875	1550	0.96	15	1550.0	0	4162.6
p5.2.x	64	1610	1610	0	128	1610	0	18	1610.0	0	4312.1
p5.2.y	64	1645	1645	0	457	1640	0.30	16	1635.5	4.63	4475.6
p5.2.z	64	1680	1680	0	0	1675	0.30	20	1670.1	0.53	4474.5
p5.3.l	64	595	595	0	31	595	0	12	595.0	0	2820.9
p5.3.m	64	650	650	0	1	650	0	12	649.4	1.62	3161.9
p5.3.n	64	755	755	0	3	755	0	15	755.0	0	3514.4
p5.3.q	64	1070	1090	1.83	521	1065	2.29	20	1065.0	0	4440.3
p5.3.t	64	1260	1270	0.79	5152	1250	1.57	20	1234.6	6.63	5300.9
p5.3.u	64	1350	1395	3.23	123	1335	4.30	22	1317.1	4.58	5558.1
p5.4.l	44	430	430	0	0	430	0	12	428.7	2.67	2746.2
p5.4.m	52	555	555	0	0	555	0	15	550.9	4.13	3384.2
p5.4.n	60	620	620	0	0	620	0	15	619.2	6.48	3749.8
p5.4.o	60	690	690	0	0	685	0.72	16	675.6	4.73	4183.1
p5.4.p	64	765	765	0	729	760	0.65	25	751.2	3.22	4629.4
p5.4.q	64	860	860	0	1	860	0	20	859.1	3.69	5075.5
p5.4.v	64	1320	1320	0	12	1320	0	28	1319.2	4.33	6731.2
p5.4.y	64	1520	1520	0	46	1500	1.32	29	1479.3	8.41	7778.3
p5.4.z	64	1620	1620	0	550	1560	3.70	30	1540.7	6.14	8027.5
Avg		1115.8	1118.3	0.19	961.5	1109.3	0.66	16.5	1102.6	3.38	4124.7

Table 2: Comparison of the event based weighted sum routing algorithm with best-known solutions provided in [20] to the standard benchmark instance for the team orienteering problem.

Instance	nodes	Benchmark				Our solution					
		best LB	best UB	Gap	Time	best	Gap	Time	Average solution	Standard deviation	Total time
p6.2.j	62	948	948	0	139	948	0	10	935.3	3.92	2630.3
p6.2.k	62	1032	1032	0	223	1032	0	8	1019.8	4.79	2738.8
p6.2.l	62	1116	1116	0	39	1110	0.54	11	1104.1	0.63	3246.5
p6.2.m	62	1188	1188	0	680	1188	0	15	1170.2	1.96	3545.6
p6.2.n	62	1260	1260	0	1	1242	1.43	17	1227.7	5.88	3609.7
p6.3.m	62	1080	1080	0	432	1080	0	12	1062.1	4.01	4330.9
Set Avg		1104.0	1104.0	0	252.3	1100.0	0.33	12.2	1086.5	3.53	3350.3
p7.2.g	87	459	459	0	44	442	3.70	13	433.4	1.35	2941.8
p7.2.h	92	521	521	0	1977	508	2.50	20	501.4	3.31	4089.4
p7.2.i	98	580	580	0	6271	559	3.62	27	548.6	1.83	4775.9
p7.2.t	100	1179	1179	0	6934	1105	6.28	62	1066.7	12.76	13695.9
p7.3.h	59	425	425	0	3	410	3.53	15	396.6	5.79	4305.9
p7.3.i	70	487	487	0	488.5	473	2.87	14	462.9	4.13	5272.2
p7.3.j	80	564	564	0	4207	548	2.84	31	532.8	6.41	7033.9
p7.3.k	91	633	633.2	0.03	1173	602	4.92	33	587.6	4.50	7896.5
p7.3.m	96	762	762	0	928	712	6.56	45	700.4	7.34	10915.7
p7.3.n	99	820	820	0	230	758	7.56	59	744.0	4.86	12765.8
p7.4.j	51	462	462	0	2	441	4.55	24	429.7	6.26	6385.5
p7.4.k	61	520	520	0	73	506	2.69	32	492.4	5.12	8083.5
p7.4.l	70	590	590	0	173	566	4.07	36	549.5	5.56	9541.7
p7.4.n	87	730	730	0	85	701	3.97	53	676.1	10.27	12753.5
p7.4.o	91	781	784.7	0.47	4434	762	2.89	73	732.1	9.09	14417.8
Set Avg		634.2	634.5	0.03	1801.5	606.2	4.17	35.7	590.3	5.91	8325.0
Avg set Avg		789.9	790.3	0.04	922.5	773.5	2.32	18.0	760.1	4.95	4767.4

Table 3: Comparison of four algorithms in problem settings with varying degrees of path-dependency, 100 nodes, 4 drones with random uniform rewards and positions in a 100 metre by 100 metre by 20 metre box.

Dynamism experiment settings				Results														
				Total reward				Solution time (seconds)				RMSE of edge traversal time predictions						
Air-resistance	Gravity	Turn penalties on	Partial observations	LH	RMLH	RMSA	PH	LH	RMLH	RMSA	PH	LH	RMLH	RMSA	PH			
Low	0	Off	Off	49.96	49.96	49.96	49.96	124	2998	305	190	0.14	-	-	0.06			
			On	49.96	49.96	49.96	49.96	157	2934	294	175	0.16	-	-	0.07			
		On	Off	47.86	49.42	48.38	48.79	110	3513	378	165	0.17	-	-	0.09			
			On	49.68	49.26	48.32	48.75	149	3658	367	164	0.16	-	-	0.09			
	9.81	Off	Off	25.29	25.47	22.47	24.12	65	3498	388	63	0.16	-	-	0.08			
			On	28.45	28.84	27.78	28.82	75	3592	383	83	0.16	-	-	0.09			
		On	Off	24.39	24.30	22.83	23.72	52	3157	387	73	0.12	-	-	0.10			
			On	28.35	28.57	24.64	27.82	79	3769	313	87	0.13	-	-	0.09			
			High	0	Off	Off	30.78	29.67	28.71	29.79	68	3491	388	87	0.07	-	-	0.05
						On	33.85	33.60	32.88	34.22	67	3214	435	99	0.07	-	-	0.05
On	Off	30.45			30.40	25.57	29.05	81	4248	430	83	0.10	-	-	0.07			
	On	33.30			33.54	33.07	33.13	87	4101	411	104	0.09	-	-	0.08			
9.81	Off	Off		6.70	6.70	6.70	6.70	11	3206	363	15	0.07	-	-	0.08			
		On		10.48	10.27	10.16	10.48	8	2840	336	15	0.06	-	-	0.06			
	On	Off		6.58	6.58	6.58	6.38	12	3268	349	17	0.07	-	-	0.09			
		On		10.30	10.27	10.27	10.27	10	3050	325	16	0.07	-	-	0.08			
Average				29.15	29.18	28.02	28.87	72	3409	366	90	0.11	-	-	0.08			

Table 4: Comparison of four algorithms in problem settings with varying degrees of path-dependency, 150 nodes, 4 drones with random uniform rewards and positions in a 100 metre by 100 metre by 40 metre box.

Dynamism experiment settings				Results											
				Total reward				Solution time (seconds)				RMSE of edge traversal time predictions			
Air-resistance	Gravity	Turn penalties on	Partial observations	LH	RMLH	RMSA	PH	LH	RMLH	RMSA	PH	LH	RMLH	RMSA	PH
Low	0	Off	Off	60.92	62.04	59.86	59.89	456	6938	693	612	0.14	-	-	0.07
			On	63.04	63.42	58.81	63.98	491	7636	688	672	0.16	-	-	0.06
		On	Off	56.89	56.26	57.05	56.51	449	8051	766	632	0.18	-	-	0.10
			On	59.14	58.14	58.44	58.64	567	7123	778	504	0.18	-	-	0.09
	9.81	Off	Off	24.04	24.17	23.96	24.36	123	4834	763	126	0.19	-	-	0.13
			On	31.24	30.76	28.49	29.31	155	7421	654	202	0.19	-	-	0.12
		On	Off	23.78	22.52	22.11	22.85	160	6488	698	81	0.17	-	-	0.13
			On	28.66	28.93	28.71	28.21	174	6917	710	168	0.17	-	-	0.14
			High	0	Off	Off	25.59	25.78	24.52	25.55	166	6947	774	179	0.07
On	30.76	31.04	29.45			29.84	132	7662	744	210	0.06	-	-	0.05	
On	Off	25.31	25.55		24.62	24.79	122	7501	820	155	0.11	-	-	0.09	
	On	30.28	31.02		29.29	30.06	170	7619	707	158	0.10	-	-	0.08	
9.81	Off	Off	1.37		1.37	1.37	1.37	4	1143	404	5	0.03	-	-	0.03
		On	4.96		4.96	4.96	4.96	4	1142	404	4	0.03	-	-	0.04
	On	Off	1.37		1.37	1.37	1.37	5	1108	408	5	0.03	-	-	0.03
		On	4.96		4.96	4.96	4.96	5	1111	409	5	0.03	-	-	0.04
		Average				29.52	29.52	28.62	29.17	199	5603	651	232	0.11	-

In the 150 node instance, all of the algorithms achieve the same solution for the four problem settings in which the level of interaction with the environment is highest. This is because, in these cases and due to high resistive forces, none of the drones have enough time to visit any nodes too far from the start and end depots.

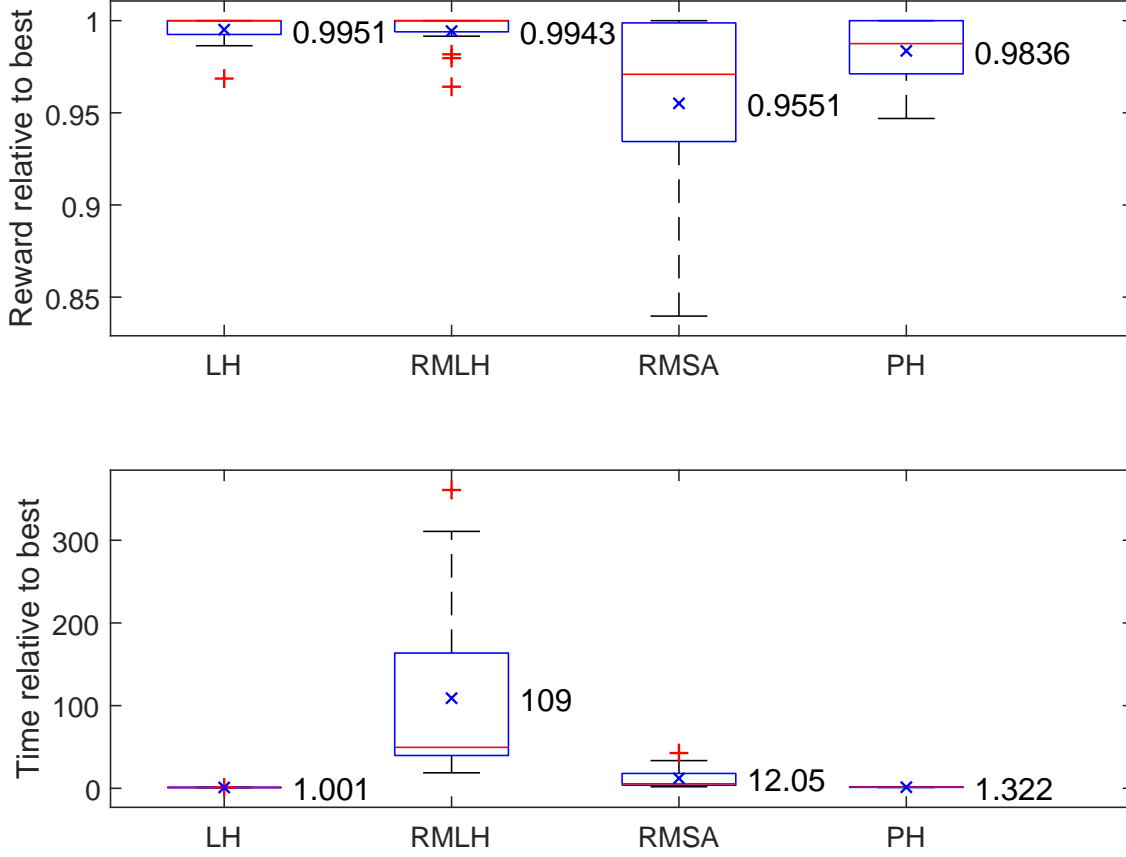


Figure 7: Visual comparison of rewards and solutions times of four algorithms relative to the maximum reward and minimum solution time respectively for each instance in Table 3.

Considering the results of Tables 3 and 4, the *LH* method attained the highest or joint highest total reward (in 17 out of 32 cases), and the second highest overall average reward, i.e.: 29.34 *LH*, 29.35 *RMLH*, 28.32 *RMSA*, and 29.02 *PH*. The *RMLH* method attained the best overall average reward, the small difference between *LH* and the *RMLH* represents the cost of using the ML module to quickly evaluate tour times. The solution times of all four algorithms increased in environments with low levels or resistive forces. This is because Algorithm 4 evaluates the efficiency attributes of every unvisited node as a candidate node every time a new next-node is required for a drone. Also, the higher the velocities the more node visits have to be evaluated using the ML module. Tables 3 and 4 show that the *LH* method attains quality solutions in short times (between 4 seconds and 491 seconds) in comparison to the *RMLH* method (1, 108 to 8, 051 seconds) and to the *RMSA* method (294 to 820 seconds). Figures 7 and 8 provide a visual comparison of the rewards and computing times achieved by each algorithm. It is interesting to note that the *RMLH* algorithm is often outperformed by the *LH* algorithm despite they both are similar algorithms –but with the latter using a relatively unrestricted access to reality-module evaluations. One explanation for this is that the integrated learning and optimisation process has the added benefit of increasing the diversity of search, a feature that is lost when no uncertainty exists in decision cost predictions (as is the case for the *RMSA* algorithm). Since the *RMLH* algorithm does already have a substantial amount of randomisation, we conclude that the *LH* algorithm gives rise to an additional beneficial search-diversification mechanism. The experiments of Tables 3 and 4 have been repeated for 6 other types of instances, also available from the previously cited website. In these instances, the node positions are arranged in different ways. The results of those experiments are qualitatively

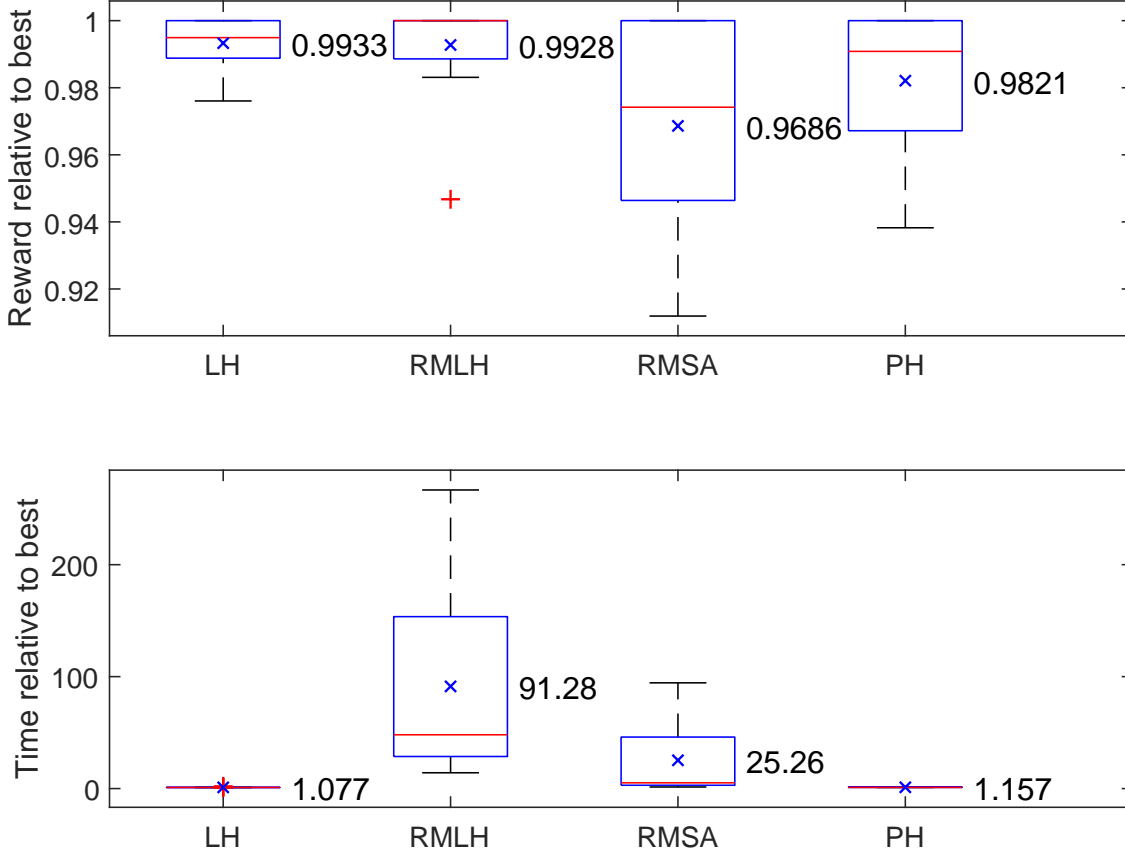


Figure 8: Visual comparison of rewards and solutions times of four algorithms relative to the maximum reward and minimum solution time respectively for each instance in Table 4.

very similar to the results of Tables 3 and 4. The associated appendix provides the results of these experiments.

10 Conclusion

In this paper, a learnheuristic approach has been proposed to solve the team orienteering problem with path-dependent travel times between each pair of targets. The problem refers to aerial drones, and edge-travel times depend upon the prior route taken by a drone due to: (i) the need to slow down for sharp turns; and (ii) the effects of gravity and air-resistance on the motion of a drone. The equations of motion require detailed numerical solutions, which are time consuming to calculate. This constitutes the main computational bottleneck. Our proposed algorithm integrates a metaheuristic with a machine learning mechanism that allows it to learn how to approximate edge costs quickly from a limited budget of reality-module evaluations. This way, our approach can significantly speed-up the computation times required to achieve high-quality solutions to such a complex problem. The algorithm also benefits from biased-randomisation techniques and different local search processes. Another finding of the work is that the learning process also helps to improve search diversity of the proposed learnheuristic. To address the learning objective of the simultaneous learning and optimisation problem, a prediction uncertainty measure was used to encourage the exploration of rarely-visited edges. A series of numerical experiments contribute to support the effectiveness of our approach.

A critical ingredient to the success of a learnheuristic algorithm lies in the correct selection of the optimisation and learning modules. In particular, it is advisable to employ problem-domain knowledge in the design of the learning component. In this work, the learning module utilised domain knowledge about the convexity of edge-

traversal times, as well as on edge final velocities as a function of the initial velocity. The interpolation method employed also benefited from an adaptive procedure for setting the vital interpolation weight parameter. This highlights the importance of using all of the available feedback from reality-module runs in order to improve the machine-learning module. In cases where the functions being predicted do not exhibit concavity or convexity, the exact details of an interpolation approach will need to be modified, ideally by exploiting any available domain specific knowledge. However it is worth noting that, as more data becomes available –and assuming a non-fractal function–, it becomes increasingly likely that the local region of the function relevant to an instance-based interpolation tends to convexity or concavity. For cases where the functions being predicted exhibit uncertainty (due to factors such as weather conditions), it may be advisable to use a more general machine learning approach such as neural networks.

Different research lines can be foreseen for future work, among them: (i) the exploration of the use of more advanced learning methods, such as multiple regression, support vector machines, random forests, or neural networks; (ii) the integration of stochastic components in the inputs (e.g., in the rewards provided by each target); (iii) the inclusion of different driving-range limits on a heterogeneous fleet of drones; and (iv) the incorporation of uncertainty due to weather conditions.

Acknowledgements

This work has been partially supported by Rhenus Freight Logistics GmbH & Co. KG and by the Spanish Ministry of Science, Innovation, and Universities (RED2018-102642-T). We acknowledge the support of the Erasmus+ Program (2019-I-ES01-KA103-062602). We also acknowledge the help of Christine Currie for initialising this project and the anonymous reviewers whose comments have greatly helped to enhance this article.

REFERENCES

- [1] L. Calvet, A. Ferrer, M. I. Gomes, A. A. Juan, and D. Masip, “Combining statistical learning with metaheuristics for the multi-depot vehicle routing problem with market segmentation,” *Computers & Industrial Engineering*, vol. 94, pp. 93–104, 2016.
- [2] L. Calvet, J. de Armas, D. Masip, and A. A. Juan, “Learnheuristics: Hybridizing metaheuristics with machine learning for optimization with dynamic inputs,” *Open Mathematics*, vol. 15, no. 1, pp. 261–280, 2017.
- [3] C. Fikar, A. A. Juan, E. Martinez, and P. Hirsch, “A discrete-event driven metaheuristic for dynamic home service routing with synchronised trip sharing,” *European Journal of Industrial Engineering*, vol. 10, no. 3, pp. 323–340, 2016.
- [4] S. Gonzalez-Martin, A. A. Juan, D. Riera, Q. Castella, R. Muñoz, and A. Perez, “Development and assessment of the sharp and randsharp algorithms for the arc routing problem,” *AI Communications*, vol. 25, no. 2, pp. 173–189, 2012.
- [5] C. L. Quintero-Araujo, A. Gruler, A. A. Juan, and J. Faulin, “Using horizontal cooperation concepts in integrated routing and facility-location decisions,” *International Transactions in Operational Research*, vol. 26, no. 2, pp. 551–576, 2019.
- [6] A. Ferrer, D. Guimarans, H. Ramalhinho, and A. A. Juan, “A brils metaheuristic for non-smooth flow-shop problems with failure-risk costs,” *Expert Systems with Applications*, vol. 44, pp. 177–186, 2016.
- [7] E. M. Gonzalez-Neira, D. Ferone, S. Hatami, and A. A. Juan, “A biased-randomized simheuristic for the distributed assembly permutation flowshop problem with stochastic processing times,” *Simulation Modelling Practice and Theory*, vol. 79, pp. 23–36, 2017.
- [8] J. De Armas, A. A. Juan, J. M. Marquès, and J. P. Pedroso, “Solving the deterministic and stochastic uncapacitated facility location problem: from a heuristic to a simheuristic,” *Journal of the Operational Research Society*, vol. 68, no. 10, pp. 1161–1176, 2017.

- [9] I. H. Witten, E. Frank, and M. A. Hall, *Data mining: Practical machine learning tools and techniques*. Burlington, USA: Elsevier, 2011.
- [10] C. Bayliss, C. S. Currie, J. A. Bennell, and A. Martinez-Sykora, "Dynamic pricing for vehicle ferries: Using packing and simulation to optimize revenues," *European Journal of Operational Research*, vol. 273, no. 1, pp. 288–304, 2019.
- [11] B. Golden, L. Levy, and R. Vohra, "The orienteering problem," *Naval Research Logistics*, vol. 34, pp. 307–318, 1987.
- [12] A. Gunawan, H. Lau, and P. Vansteenwegen, "Orienteering problem: A survey of recent variants, solution approaches and applications," *European Journal of Operational Research*, vol. 255, pp. 315–332, 2016.
- [13] W. Souffriau, P. Vansteenwegen, J. Vertommen, G. Vanden Berghe, and D. Van Oudheusden, "A personalised tourist trip design algorithm for mobile tourist guides," *Applied Artificial Intelligence*, vol. 22, no. 10, pp. 964–985, 2008.
- [14] C. Verbeeck, K. Sörensen, E.-H. Aghezzaf, and P. Vansteenwegen, "A fast solution method for the time-dependent orienteering problem," *European Journal of Operational Research*, vol. 236, pp. 419–432, 2014.
- [15] I.-M. Chao, B. Golden, and E. Wasil, "The team orienteering problem," *European Journal of Operational Research*, vol. 88, pp. 464–474, 1996.
- [16] S.-W. Lin and V. F. Yu, "A simulated annealing heuristic for the team orienteering problem with time windows," *European Journal of Operational Research*, vol. 217, pp. 94–107, 2012.
- [17] V. F. Yu, P. Jewpanya, C.-J. Ting, and A. P. Redi, "Two-level particle swarm optimization for the multi-modal team orienteering problem with time windows," *Applied Soft Computing*, vol. 61, pp. 1022–1040, 2017.
- [18] S. Butt and D. Ryan, "An optimal solution procedure for the multiple tour maximum collection problem using column generation," *Computers and Operations Research*, vol. 26, pp. 427–441, 1999.
- [19] M. Keshtkaran, K. Ziarati, A. Bettinelli, and D. Vigo, "Enhanced exact solution methods for the team orienteering problem," *International Journal of Production Research*, vol. 54, pp. 591–601, 2015.
- [20] R. El-Hajj, D.-C. Dang, and A. Moukrim, "Solving the team orienteering problem with cutting planes," *Computers and Operations Research*, vol. 74, pp. 21–30, 2016.
- [21] D.-C. Dang, R. N. Guibadj, and A. Moukrim, "An effective PSO-inspired algorithm for the team orienteering problem," *European Journal of Operational Research*, vol. 229, no. 2, pp. 332–344, 2013.
- [22] V. Campos, R. Martí, J. Sánchez-Oro, and A. Duarte, "GRASP with path relinking for the orienteering problem," *Journal of the Operational Research Society*, vol. 65, no. 12, pp. 1800–1813, 2014.
- [23] L. Ke, L. Zhai, J. Li, and F. T. Chan, "Pareto mimic algorithm: An approach to the team orienteering problem," *Omega*, vol. 61, pp. 155–166, 2016.
- [24] S. Lin, "Solving the team orienteering problem using effective multi-start simulated annealing," *Applied Soft Computing*, vol. 13, pp. 1064–1073, 2013.
- [25] E. Tsakirakis, M. Marinaki, Y. Marinakis, and N. Matsatsinis, "A similarity hybrid harmony search algorithm for the team orienteering problem," *Applied Soft Computing*, vol. 75, pp. 130–147, 2019.
- [26] J. Ferreira, A. Quintas, and J. Oliveira, "Solving the team orienteering problem: developing a solution tool using a genetic algorithm approach," in *Soft computing in industrial applications. Advances in Intelligent Systems and Computing*: 223. Springer, 2014, pp. 365–375.
- [27] G. Kobeaga, M. Merino, and J. A. Lozano, "An efficient evolutionary algorithm for the orienteering problem," *Computers and Operations Research*, vol. 90, pp. 42–59, 2018.
- [28] W. Hu, M. Fathi, and P. M. Pardalos, "A multi-objective evolutionary algorithm based on decomposition and constraint programming for the multi-objective team orienteering problem with time windows," *Applied Soft Computing*, vol. 73, pp. 383–393, 2018.

- [29] T. Ilhan, S. Iravani, and M. Daskin, "The orienteering problem with stochastic profits," *IIE Transactions*, vol. 40, pp. 406–421, 2008.
- [30] J. Panadero, C. Currie, A. A. Juan, and C. Bayliss, "Maximizing reward from a team of surveillance drones under uncertainty conditions: a simheuristic approach," *European Journal of Industrial Engineering*, 2020.
- [31] S. M. Shavarani, M. G. Nejad, F. Rismanchian, and G. Izbirak, "Application of hierarchical facility location problem for optimization of a drone delivery system: a case study of amazon prime air in the city of san francisco," *The International Journal of Advanced Manufacturing Technology*, vol. 95, no. 9-12, pp. 3141–3153, 2018.
- [32] A. Troudi, S.-A. Addouche, S. Dellagi, and A. El Mhamedi, "Logistics support approach for drone delivery fleet," in *International Conference on Smart Cities*. Springer, 2017, pp. 86–96.
- [33] D. Moormann, "DHL parcelcopter research flight campaign 2014 for emergency delivery of medication," 2015, in *Proceedings of the ICAO RPAS Symposium*, 2015.
- [34] M. Roberts, D. Dey, A. Truong, S. Sinha, S. Shah, A. Kapoor, P. Hanrahan, and N. Joshi, "Submodular trajectory optimization for aerial 3D scanning," in *International Conference on Computer Vision (ICCV)*. Computer Vision Foundation, 2017, pp. 5324–5333.
- [35] W. P. Coutinho, J. Fliege, and M. Battarra, "Glider routing and trajectory optimisation in disaster assessment," *European Journal of Operational Research*, vol. 274, pp. 1138–1154, 2019.
- [36] L. Zhen, M. Li, G. Laporte, and W. Wang, "A vehicle routing problem arising in unmanned aerial monitoring," *Computers and Operations Research*, vol. 105, pp. 1–11, 2019.
- [37] A. Otto, N. Agatz, J. Campbell, B. Golden, and E. Pesch, "Optimization approaches for civil applications of unmanned aerial vehicles (UAVs) or aerial drones: A survey," *Networks*, vol. 72, pp. 411–458, 2018.
- [38] C. C. Murray and A. G. Chu, "The flying sidekick traveling salesman problem: Optimization of drone-assisted parcel delivery," *Transportation Research Part C*, vol. 54, pp. 86–109, 2015.
- [39] S. Poikonen, X. Wang, and B. Golden, "The vehicle routing problem with drones: Extended models and connections," *Networks*, vol. 70, no. 1, pp. 34–43, 2017.
- [40] K. Sundar and S. Rathinam, "Algorithms for routing an unmanned aerial vehicle in the presence of refueling depots," *IEEE Transactions on Automation Science and Engineering*, vol. 11, no. 1, pp. 1–9, 2013.
- [41] S. Venkatachalam, K. Sundar, and S. Rathinam, "A two-stage approach for routing multiple unmanned aerial vehicles with stochastic fuel consumption," *Sensors*, vol. 18, no. 11, p. 3756, 2018.
- [42] K. Dorling, J. Heinrichs, G. G. Messier, and S. Magierowski, "Vehicle routing problems for drone delivery," *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, vol. 47, no. 1, pp. 70–85, 2017.
- [43] S. J. Kim, G. J. Lim, and J. Cho, "Drone flight scheduling under uncertainty on battery duration and air temperature," *Computers and Industrial Engineering*, vol. 117, pp. 291–302, 2018.
- [44] L. E. Dubins, "On curves of minimal length with a constraint on average curvature, and with prescribed initial and terminal positions and tangents," *American Journal of Mathematics*, vol. 79, no. 3, pp. 497–516, 1957.
- [45] J. L. Ny, E. Frazzoli, and E. Feron, "The curvature-constrained traveling salesman problem for high point densities," in *Proceedings of the 46th IEEE Conference on Decision and Control*. New Orleans, USA: IEEE Press, 2007, pp. 5985–5990.
- [46] J. T. Isaacs and J. P. Hespanha, "Dubins traveling salesman problem with neighborhoods: A graph-based approach," *Algorithms*, vol. 6, pp. 84–99, 2013.
- [47] K. Savla, E. Frazzoli, and F. Bullo, "Traveling salesperson problems for the dubins vehicle," *IEEE Transaction on Automatic Control*, vol. 53, no. 6, pp. 1378–1391, 2008.

- [48] L. Babel, "Curvature-constrained traveling salesman tours for aerial surveillance in scenarios with obstacles," *European Journal of Operational Research*, vol. 262, no. 11, pp. 335–346, 2017.
- [49] A. Zulu and S. John, "A review of control algorithms for autonomous quadrotors," *Open Journal of Applied Sciences*, vol. 4, pp. 547–556, 2014.
- [50] Y. Jin, "A comprehensive survey of fitness approximation in evolutionary computation," *Soft Computing*, vol. 5, pp. 3–12, 2005.
- [51] Y. Jin and B. Sendhoff, "Reducing fitness evaluations using clustering techniques and neural network ensembles," in *Genetic and Evolutionary Computation Conference*, ser. LNCS3102, K. D. et al., Ed. Springer, 2004, pp. 688–699.
- [52] A. Brownlee, O. Regnier-Coudert, J. McCall, and S. Massie, "Using a markov network as a surrogate fitness function in a genetic algorithm," in *Proceedings of the IEEE Congress on Evolutionary Computation*. IEEE Press, 2010, pp. 1–8.
- [53] D. Lim, Y. Jin, Y.-S. Ong, and B. Sendhoff, "Generalizing surrogate-assisted evolutionary computation," *IEEE Transactions on Evolutionary Computation*, vol. 14, no. 3, pp. 329–355, 2010.
- [54] S. Yang, Q. H. Liu, J. Lu, S.L.Ho, G. Ni, P. Ni, and S. Xiong, "Application of support vector machines to accelerate the solution speed of metaheuristic algorithms," *IEEE Transactions on Magnetics*, vol. 45, no. 3, pp. 1502–1505, 2009.
- [55] L. P. Kaelbling, M. L. Littman, and A. W. Moore, "Reinforcement learning: A survey," *Journal of Artificial Intelligence Research*, vol. 4, pp. 237–285, 1996.
- [56] E. A. Feinberg and A. Shwartz, *Handbook of Markov decision processes: methods and applications*. Springer Science & Business Media, 2012, vol. 40.
- [57] R. Bellman, "Dynamic programming," *Science*, vol. 153, no. 3731, pp. 34–37, 1966.
- [58] H. Robbins, "Some aspects of the sequential design of experiments," *Bulletin of the Americal Mathematical Society*, vol. 58, pp. 527–535, 1952.
- [59] M. N. Katehakis and A. F. V. Jr, "The multi-armed bandit problem: Decomposition and computation," *Mathematics of Operations Research*, vol. 12, no. 2, pp. 185–376, 1987.
- [60] J. Gittins, K. Glazebrook, and R. Weber, *Multi-armed bandit allocation indices*. John Wiley & Sons, 2011.
- [61] R. F. Stengel, *Flight dynamics*. Princeton University Press, 2015.
- [62] P. A. Tipler and G. Mosca, *Physics for scientists and engineers*. Macmillan, 2007.
- [63] A. French and M. G. Ebison, *Introduction to classical mechanics*. Springer Science & Business Media, 2012.
- [64] K. A. Stroud and D. J. Booth, *Advanced engineering mathematics*. London: Palgrave Macmillan, 2003.
- [65] J. J. Hopfield, "Neural networks and physical systems with emergent collective computational abilities," *Proceedings of the National Academy of Sciences of the United States of America*, vol. 79, no. 8, pp. 2554–2558, 1982.
- [66] J. Gareth, *An introduction to statistical learning: with applications in R*. Springer Verlag, 2010.
- [67] P. Vansteenwegen, W. Souffriau, and D. V. Oudheusden, "The orienteering problem: A survey," *European Journal of Operational Research*, vol. 209, pp. 1–10, 2011.
- [68] O. Dominguez, A. A. Juan, I. de la Nuez, and D. Ouelhadj, "An ils-biased randomization algorithm for the two-dimensional loading hfvpr with sequential loading and items rotation," *Journal of the Operational Research Society*, vol. 67, no. 1, pp. 37–53, 2016.
- [69] C. L. Quintero-Araujo, J. P. Caballero-Villalobos, A. A. Juan, and J. R. Montoya-Torres, "A biased-randomized metaheuristic for the capacitated location routing problem," *International Transactions in Operational Research*, vol. 24, no. 5, pp. 1079–1098, 2017.

- [70] S. Kirkpatrick, C. D. Gelatt, Jr, and M. P. Vecchi, "Optimization by simulated annealing," *Science*, vol. 220, no. 4598, pp. 671–680, May 1983.
- [71] E. Aarts and J. K. Lenstra, *Local search and combinatorial optimization*. Chichester, New York, Weinheim, Brisbane, Singapore and Toronto: John Wiley and Sons, 1997.
- [72] D. Ferone, A. Gruler, P. Festa, and A. A. Juan, "Enhancing and extending the classical grasp framework with biased randomisation and simulation," *Journal of the Operational Research Society*, vol. 70, no. 8, pp. 1362–1375, 2019.

Summary of the notation

B	: Set of drones
n_i	: Node i
N	: Set of nodes
M	: Set of nodes excluding start and end depot nodes
$p(n)$: Spatial coordinate of node i
$s(n)$: Reward for a direct visit of node n
T_{max}	: Maximum time available to drones for collecting rewards and returning to the end depot before rewards collected by a drone are lost
H	: Cruising height of a drone
W	: Node width (assumed to be spherical in shape)
l_{jk}	: distance between node j and node k
q_{kj}	: Partial reward available for an observation of node k from node j if node j is the closest node visited directly by a drone
G	: Graph composed of the set of nodes N and set of edges E
E	: Set of edges
Y_{bk}	: The k^{th} node visited by drone b
$TourTime(Y_b)$: Function returning the actual tour time of drone b 's tour Y_b
β_{ij}	: Gradient, with respect to horizontal ground, of the straight line path from node i to node j
τ_b	: Actual time required for drone b to complete their assigned route Y_b
$h(u, \beta_{ij}, l_{ij})$: Function returning the true traversal time and final velocity for a drone traversing the edge from node i to node j starting from an initial velocity of u , with a gradient β_{ij} and length l_{ij}
π_{ijk}	: Multiplicative turn penalty applied to a drone's speed upon reaching node j starting from node i with node k the next node after node j in the drone's tour
e_{ij}	: Directed edge starting at node i and ending at node j
u	: A given drone's initial velocity (predicted or actual depending upon the context) when starting to traverse a given edge
v	: A given drone's final velocity (predicted or actual depending upon the context) upon completion of a given edge traversal
t	: The time required for a drone to complete the traversal of a given edge
$reward$: The total reward (predicted or actual depending upon the context) associated with a given set of drone tours $Y_b \forall b \in B$
U	: Set of non-visited nodes at a given stage of solution construction or solution evaluation
\mathcal{B}	: Set of drones which have completed their assigned tours (predicted or actual depending upon the context)
b	: A drone in the set \mathcal{B}
v_{bki}	: Binary integer decision variable which indicates with a value of 1 whether edge e_{ij} is the k^{th} edge in drone b 's tour
F	: Thrust force available to a drone
α	: Air-resistance coefficient
ρ	: Density of air
C_D	: Drag coefficient
A	: Cross-sectional area of a drone
m	: Mass of a drone
g	: Gravitational acceleration
r_x	: Resultant force upon drone in the direction parallel to horizontal ground
r_y	: Resultant for upon a drone in the direction parallel to the vertical
f_x	: horizontal component of the thrust force of a drone
f_z	: Vertical component of the thrust force of a drone
δ	: Time interval size used in the numerical solution to the equations of motion governing a drones motion throughout their assigned tour
$position$: The distance that a drone has travelled along a given edge
s	: The current speed of a drone on a given edge
$\phi(s, \beta, g, \alpha)$: Function returning the current acceleration of a drone with a current velocity of s along an edge with a gradient of β , under a gravitation acceleration of g and air-resistance with air-resistance coefficient α
θ_{ijk}	: The angle between edges e_{ij} and e_{jk}
$h'_{ij}(u)$: Function returning the machine learning based traversal time and final velocity predictions for a drone traversing the edge from node i to node j starting from an initial velocity of u
D^{ij}	: Set of data collected regarding actual drone traversals of the directed edge e_{ij}
d_k^{ij}	: The data point in the set D^{ij} with the k^{th} lowest initial velocity
$u(d_k^{ij})$: Function returning the initial velocity of the k^{th} data point collected for traversals of edge e_{ij}
$v(d_k^{ij})$: Function returning the final velocity of the k^{th} data point collected for traversals of edge e_{ij}
$t(d_k^{ij})$: Function returning the edge traversal time of the k^{th} data point collected for traversals of edge e_{ij}
u'	: Initial velocity for an edge traversal for which final velocity (v') and edge traversal time (t') predictions are required
v'	: Predicted final velocity for an edge traversal
t'	: Predicted traversal time for an edge traversal
\underline{d}	: The data point for a edge traversal with the greatest initial velocity lower than or equal to u'
\overline{d}	: The data point for an edge traversal with the lowest initial velocity greater than u'
ψ	: Equal to $u(\overline{d}) - u(\underline{d})$, i.e., the difference between the initial velocities of the data points with the nearest initial velocities above and below u'
\bar{c}	: Equal to $ u' - u(\overline{d})$, i.e, the difference between the initial velocity u' and $u(\overline{d})$
\underline{c}	: Equal to $u(\underline{d}) - u' $, i.e, the difference between the initial velocity u' and $u(\underline{d})$
\hat{t}	: The true value for an edge-traversal time that is currently being predicted
\hat{v}	: The true value for a final velocity that is currently being predicted
$g_t(d_k)$: The gradient of edge traversal time as a function of initial velocity at the initial velocity corresponding to data point d_k for a given edge
$g_v(d_k)$: The gradient of final velocity as a function of initial velocity at the initial velocity corresponding to data point d_k for a given edge

$L_{t(v)}$:	Linearly interpolated prediction of an edge traversal time (final velocity)
$G_{t(v)}$:	Gradient based extrapolation of an edge traversal time (final velocity)
γ_t	:	The weight given to the linearly interpolated edge traversal time, a weight of $(1 - \gamma_t)$ is given to the gradient based edge traversal time prediction
γ_v	:	The weight given to the linearly interpolated final velocity, a weight of $(1 - \gamma_v)$ is given to the gradient based final velocity prediction
$\chi_{t(v)}$:	Interpolation weight learning rate
\mathfrak{B}	:	List of drones sorting in increasing order of last assigned node arrival time (event based constructive drone-routing algorithm)
C	:	List of candidate nodes that a drone can be assigned to next
$score(n)$:	Attractiveness score associated with selecting node n as the next node that a drone should visit
w_{bi}	:	The weight given to efficiency attribute i of drone b
λ_i	:	Efficiency attribute i score for a given drone visiting a given node next
$nextArrivalTime(b)$:	A function that returns the estimated time at which drone b arrives at their last assigned node
μ	:	Edge-traversal time prediction uncertainty efficiency attribute weight exponential decay scheme parameter
ϵ	:	Tolerance parameter for preventing ties when calculating minimum efficiency attribute weight changes that change a drone routing decision in the event-based constructive drone-routing algorithm
Δw_{bj}^i	:	The minimum change to the efficiency attribute weight j for drone b that changes the highest scoring candidate node to node i
\mathfrak{N}	:	Set of local search neighbourhoods
n_i	:	Local search neighbourhood i
$P_i^{local\ search}$:	The probability that the i^{th} local search neighbourhood in the current set of local search neighbourhoods is selected as the local search neighbourhood in the current iteration of the multi-start local search algorithm
Z	:	List of strong candidate efficiency attribute weight matrices generated throughout the multi-start local search algorithm
ω	:	Biased randomisation greediness parameter
LH	:	Learnheuristic
$RMLH$:	Reality-module-only learnheuristic
$RMSA$:	Reality module simulated annealing
PH	:	A priori learning heuristic

Table 5: Comparison of four algorithms in instance p12r3D with varying degrees of path-dependency.

Dynamism experiment settings				Results													
				Total reward				Solution time (seconds)				RMSE of edge traversal time predictions					
Air-resistance	Gravity	Turn penalties on	Partial observations	LH	RMLH	RMSA	PH	LH	RMLH	RMSA	PH	LH	RMLH	RMSA	PH		
Low	0	Off	Off	285.00	285.00	285.00	285.00	5	236	29	8	0.16	-	-	0.06		
			On	285.00	285.00	285.00	285.00	6	241	29	9	0.17	-	-	0.06		
		On	Off	285.00	285.00	285.00	285.00	6	398	40	9	0.10	-	-	0.08		
			On	285.00	285.00	285.00	285.00	7	408	39	10	0.11	-	-	0.09		
	9.81	Off	Off	270.00	270.00	280.00	260.00	6	378	41	7	0.10	-	-	0.07		
			On	276.89	272.25	268.45	276.89	6	394	40	8	0.10	-	-	0.07		
		On	Off	240.00	240.00	225.00	240.00	5	370	46	7	0.07	-	-	0.07		
			On	251.85	251.55	247.47	251.73	6	446	44	8	0.08	-	-	0.07		
		High	0	Off	Off	235.00	235.00	230.00	235.00	3	437	47	5	0.07	-	-	0.05
					On	246.40	250.73	250.73	246.74	4	448	48	6	0.06	-	-	0.05
On	Off			220.00	215.00	210.00	225.00	4	513	52	7	0.06	-	-	0.05		
	On			231.77	233.24	226.44	233.90	5	487	53	8	0.06	-	-	0.05		
9.81	Off		Off	100.00	95.00	100.00	100.00	2	487	61	3	0.07	-	-	0.07		
			On	127.90	131.99	126.75	126.75	2	509	60	3	0.08	-	-	0.06		
	On		Off	95.00	100.00	95.00	100.00	1	275	60	2	0.07	-	-	0.07		
			On	123.88	127.97	127.97	127.97	2	468	63	3	0.06	-	-	0.06		
Average				222.42	222.67	220.49	222.75	4	406	47	6	0.09	-	-	0.06		

Table 6: Comparison of four algorithms in instance p33t3D with varying degrees of path-dependency.

Dynamism experiment settings				Results												
				Total reward				Solution time (seconds)				RMSE of edge traversal time predictions				
Air-resistance	Gravity	Turn penalties on	Partial observations	LH	RMLH	RMSA	PH	LH	RMLH	RMSA	PH	LH	RMLH	RMSA	PH	
Low	0	Off	Off	800.00	800.00	800.00	800.00	6	225	27	9	0.19	-	-	0.06	
			On	800.00	800.00	800.00	800.00	6	220	26	9	0.19	-	-	0.07	
		On	Off	800.00	800.00	800.00	800.00	7	375	42	11	0.12	-	-	0.11	
			On	800.00	800.00	800.00	800.00	8	406	41	10	0.13	-	-	0.11	
		9.81	Off	Off	800.00	800.00	800.00	800.00	7	359	37	9	0.11	-	-	0.06
				On	800.00	800.00	800.00	800.00	7	365	37	10	0.11	-	-	0.06
			On	Off	780.00	770.00	760.00	780.00	7	449	45	8	0.08	-	-	0.07
				On	794.20	785.70	761.67	766.53	7	486	47	10	0.08	-	-	0.07
	High	0	Off	Off	760.00	750.00	760.00	760.00	5	459	47	7	0.07	-	-	0.05
				On	764.48	764.48	764.48	764.48	5	432	44	7	0.08	-	-	0.05
On			Off	710.00	710.00	710.00	710.00	6	574	54	9	0.08	-	-	0.06	
			On	720.94	720.94	720.94	720.94	7	547	55	10	0.08	-	-	0.06	
9.81			Off	Off	400.00	400.00	390.00	380.00	4	587	58	6	0.08	-	-	0.07
				On	451.69	451.69	424.78	451.69	4	606	60	6	0.08	-	-	0.07
			On	Off	370.00	400.00	390.00	400.00	4	614	64	6	0.08	-	-	0.07
				On	420.87	450.85	426.78	442.49	4	674	64	6	0.07	-	-	0.07
Average				685.76	687.73	681.79	686.01	6	461	47	8	0.10	-	-	0.07	

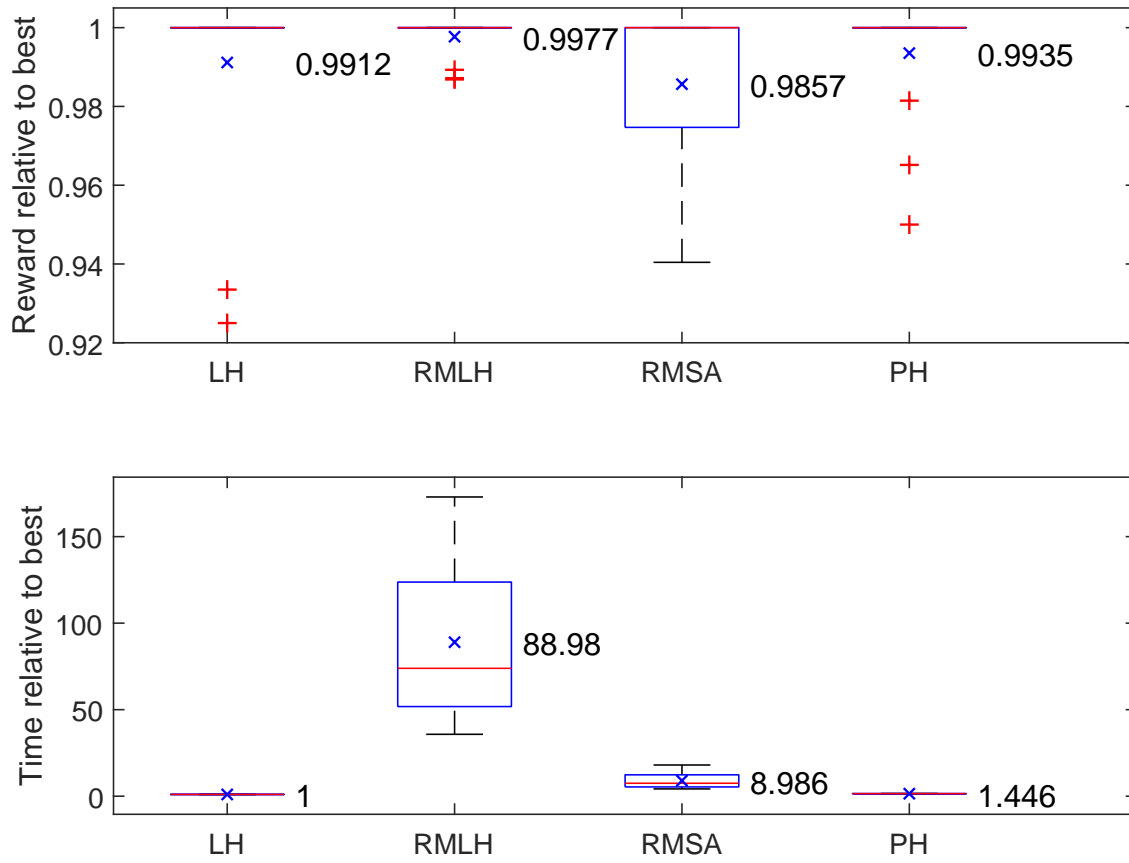


Figure 10: Visual comparison of rewards and solutions times of four algorithms relative to the maximum reward and minimum solution time respectively for each instance in Table 6.

Table 7: Comparison of four algorithms in instance p44k3D with varying degrees of path-dependency.

Dynamism experiment settings				Results												
				Total reward				Solution time (seconds)				RMSE of edge traversal time predictions				
Air-resistance	Gravity	Turn penalties on	Partial observations	LH	RMLH	RMSA	PH	LH	RMLH	RMSA	PH	LH	RMLH	RMSA	PH	
Low	0	Off	Off	1306.00	1306.00	1306.00	1306.00	138	3268	338	185	0.21	-	-	0.06	
			On	1306.00	1306.00	1306.00	1306.00	174	3451	331	189	0.20	-	-	0.09	
		On	Off	1301.00	1306.00	1306.00	1306.00	135	4700	409	149	0.15	-	-	0.11	
			On	1301.08	1306.00	1306.00	1306.00	148	4586	424	148	0.14	-	-	0.11	
		9.81	Off	Off	979.00	987.00	847.00	954.00	103	5107	430	123	0.13	-	-	0.08
				On	1049.14	1060.14	1056.80	1022.62	115	4554	445	113	0.13	-	-	0.08
			On	Off	922.00	930.00	944.00	886.00	99	4886	458	93	0.10	-	-	0.10
				On	989.00	1001.57	953.50	973.42	106	4337	419	118	0.10	-	-	0.10
	High	0	Off	Off	946.00	946.00	928.00	970.00	86	4962	502	108	0.08	-	-	0.05
				On	1005.07	1012.19	986.18	1001.37	108	5067	492	100	0.09	-	-	0.05
On			Off	913.00	920.00	870.00	912.00	90	4772	504	112	0.08	-	-	0.08	
			On	1007.99	962.52	958.84	1008.38	96	5383	530	101	0.09	-	-	0.07	
9.81		Off	Off	354.00	356.00	361.00	337.00	23	4194	467	38	0.09	-	-	0.09	
			On	483.54	469.93	451.02	465.10	20	4936	468	34	0.08	-	-	0.09	
		On	Off	377.00	347.00	347.00	350.00	29	4106	475	35	0.09	-	-	0.10	
			On	484.35	478.56	452.02	479.28	28	4437	479	36	0.08	-	-	0.10	
Average				920.26	918.43	898.71	911.45	94	4547	448	105	0.12	-	-	0.09	

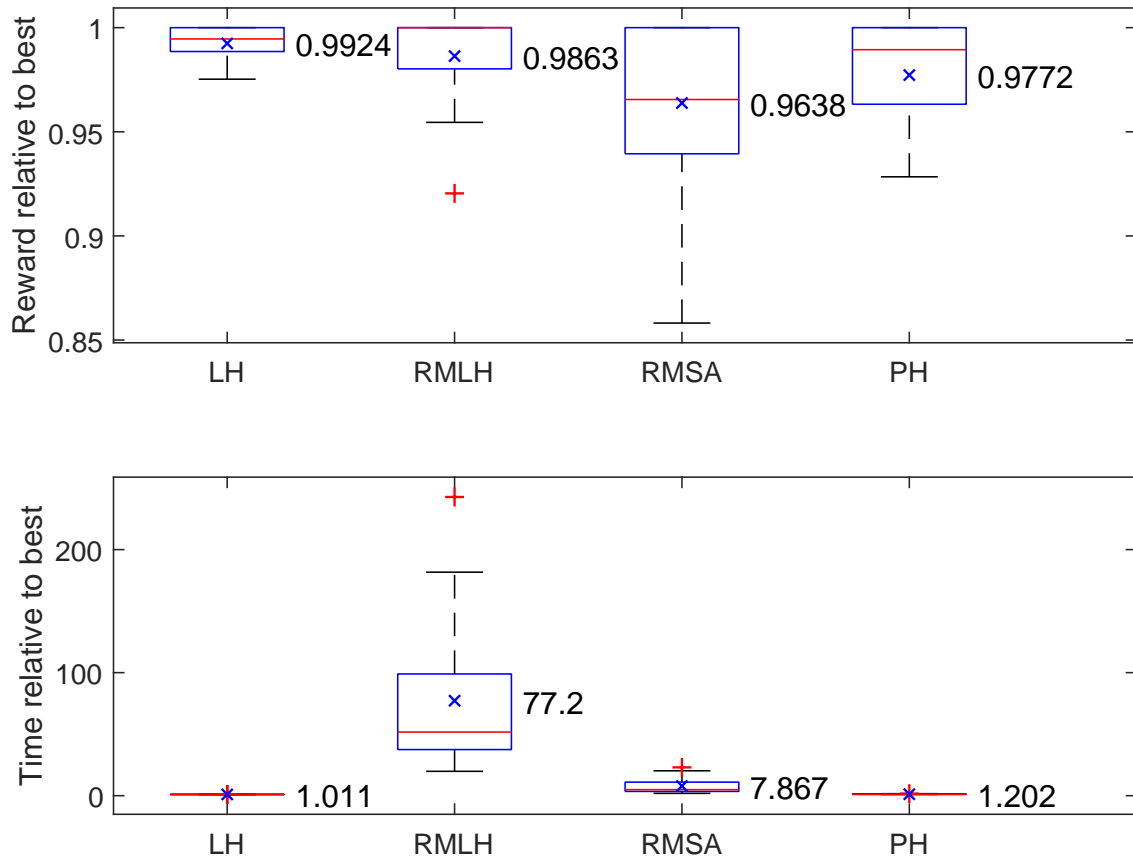


Figure 11: Visual comparison of rewards and solutions times of four algorithms relative to the maximum reward and minimum solution time respectively for each instance in Table 7.

Table 8: Comparison of four algorithms in instance p54z3D with varying degrees of path-dependency.

Dynamism experiment settings				Results											
				Total reward				Solution time (seconds)				RMSE of edge traversal time predictions			
Air-resistance	Gravity	Turn penalties on	Partial observations	LH	RMLH	RMSA	PH	LH	RMLH	RMSA	PH	LH	RMLH	RMSA	PH
Low	0	Off	Off	1570.00	1570.00	1490.00	1570.00	32	1441	137	45	0.13	-	-	0.07
			On	1542.07	1549.70	1295.00	1571.84	31	1446	130	48	0.14	-	-	0.08
		On	Off	1485.00	1395.00	1385.00	1425.00	28	2079	192	36	0.10	-	-	0.08
			On	1457.91	1446.19	1423.71	1406.12	31	2035	202	37	0.11	-	-	0.08
	9.81	Off	Off	690.00	680.00	595.00	690.00	22	1549	157	26	0.09	-	-	0.07
			On	793.68	788.63	770.14	800.85	22	1664	160	28	0.08	-	-	0.07
		On	Off	640.00	650.00	640.00	595.00	18	1793	192	23	0.08	-	-	0.08
			On	726.40	742.67	718.22	733.79	21	1693	170	25	0.09	-	-	0.08
High	0	Off	Off	515.00	515.00	515.00	515.00	13	1730	176	16	0.06	-	-	0.04
			On	611.63	611.63	611.63	617.47	13	1585	178	19	0.06	-	-	0.04
		On	Off	420.00	470.00	465.00	460.00	11	1916	192	16	0.07	-	-	0.07
			On	554.23	565.42	576.16	585.73	12	1453	203	17	0.07	-	-	0.06
	9.81	Off	Off	155.00	150.00	165.00	165.00	4	1561	187	4	0.07	-	-	0.07
			On	275.04	289.68	275.96	277.22	7	1834	187	9	0.06	-	-	0.06
		On	Off	145.00	150.00	145.00	145.00	4	1750	210	7	0.05	-	-	0.07
			On	254.10	259.12	257.61	260.07	6	1985	204	8	0.06	-	-	0.07
Average				739.69	739.56	708.03	738.63	17	1720	180	23	0.08	-	-	0.07

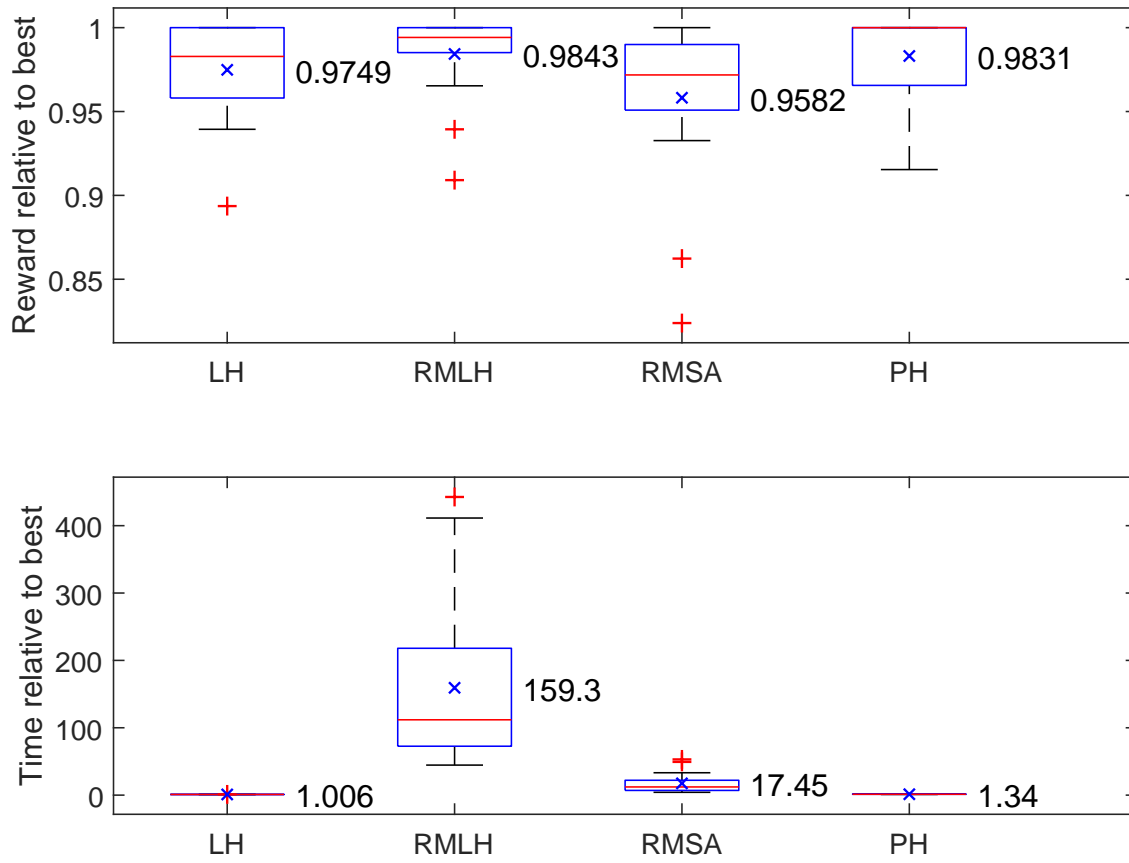


Figure 12: Visual comparison of rewards and solutions times of four algorithms relative to the maximum reward and minimum solution time respectively for each instance in Table 8.

Table 9: Comparison of four algorithms in instance p63m3D with varying degrees of path-dependency.

Dynamism experiment settings				Results													
				Total reward				Solution time (seconds)				RMSE of edge traversal time predictions					
Air-resistance	Gravity	Turn penalties on	Partial observations	LH	RMLH	RMSA	PH	LH	RMLH	RMSA	PH	LH	RMLH	RMSA	PH		
Low	0	Off	Off	1344.00	1344.00	1344.00	1344.00	33	1225	145	45	0.29	-	-	0.07		
			On	1344.00	1344.00	1344.00	1344.00	38	1229	146	49	0.27	-	-	0.07		
		On	Off	1344.00	1344.00	1344.00	1344.00	44	1752	191	44	0.18	-	-	0.09		
			On	1344.00	1344.00	1344.00	1344.00	46	1730	194	45	0.16	-	-	0.09		
	9.81	Off	Off	1326.00	1314.00	1278.00	1326.00	36	2146	209	46	0.10	-	-	0.07		
			On	1314.85	1323.92	1305.85	1327.71	37	2168	211	50	0.11	-	-	0.07		
		On	Off	1260.00	1236.00	1218.00	1236.00	33	2249	219	42	0.09	-	-	0.07		
			On	1275.92	1254.16	1228.46	1254.23	36	2267	215	44	0.09	-	-	0.07		
		High	0	Off	Off	1320.00	1320.00	1308.00	1320.00	29	2574	245	37	0.07	-	-	0.06
					On	1290.13	1281.55	1289.07	1299.91	30	2532	244	42	0.08	-	-	0.06
On	Off			1320.00	1284.00	1272.00	1290.00	30	2728	262	39	0.08	-	-	0.06		
	On			1303.22	1290.13	1267.07	1271.74	32	2753	260	39	0.08	-	-	0.06		
9.81	Off		Off	156.00	186.00	180.00	168.00	2	825	206	7	0.04	-	-	0.05		
			On	255.34	293.15	284.42	296.70	4	1038	194	5	0.04	-	-	0.05		
	On		Off	156.00	186.00	186.00	186.00	5	886	197	8	0.04	-	-	0.05		
			On	255.34	301.34	293.15	296.70	3	1798	194	8	0.04	-	-	0.05		
Average				1038.05	1040.39	1030.38	1040.56	27	1869	208	34	0.11	-	-	0.06		

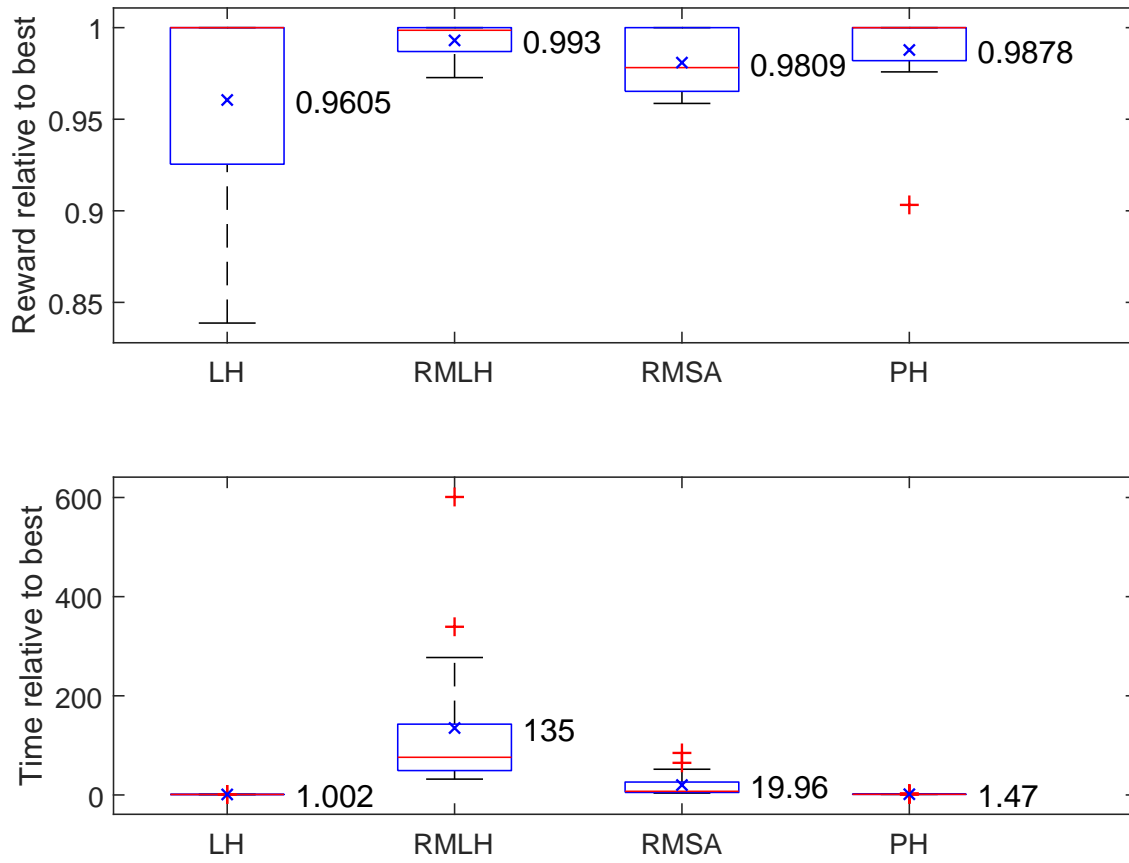


Figure 13: Visual comparison of rewards and solutions times of four algorithms relative to the maximum reward and minimum solution time respectively for each instance in Table 9.

Table 10: Comparison of four algorithms in instance p74o3D with varying degrees of path-dependency.

Dynamism experiment settings				Results											
				Total reward				Solution time (seconds)				RMSE of edge traversal time predictions			
Air-resistance	Gravity	Turn penalties on	Partial observations	LH	RMLH	RMSA	PH	LH	RMLH	RMSA	PH	LH	RMLH	RMSA	PH
Low	0	Off	Off	1436.00	1450.00	1423.00	1446.00	153	2681	266	153	0.15	-	-	0.06
			On	1420.10	1437.33	1441.42	1437.44	140	2411	260	151	0.16	-	-	0.06
		On	Off	1315.00	1338.00	1313.00	1332.00	116	4130	395	124	0.12	-	-	0.10
			On	1342.49	1337.64	1274.99	1341.28	124	4101	371	124	0.13	-	-	0.09
	9.81	Off	Off	830.00	830.00	701.00	818.00	71	3023	299	68	0.11	-	-	0.08
			On	938.09	943.22	943.21	928.62	69	3026	291	69	0.13	-	-	0.08
		On	Off	759.00	755.00	729.00	718.00	71	2576	298	67	0.09	-	-	0.10
			On	910.51	882.94	874.80	866.95	80	3195	302	74	0.09	-	-	0.09
High	0	Off	Off	842.00	821.00	815.00	839.00	74	4032	388	71	0.06	-	-	0.05
			On	952.25	951.09	943.53	956.54	63	3832	352	79	0.07	-	-	0.05
		On	Off	812.00	808.00	807.00	832.00	68	4308	417	56	0.08	-	-	0.07
			On	952.06	948.59	924.06	946.92	68	4434	373	74	0.08	-	-	0.08
	9.81	Off	Off	301.00	291.00	280.00	294.00	13	3117	275	18	0.07	-	-	0.08
			On	444.21	444.21	435.64	424.93	21	2930	286	23	0.06	-	-	0.08
		On	Off	289.00	289.00	239.00	279.00	13	1751	281	17	0.05	-	-	0.10
			On	405.11	413.47	409.13	411.41	19	2263	277	17	0.05	-	-	0.08
Average				871.8	871.28	847.11	867.01	73	3238	321	74	0.09	-	-	0.08

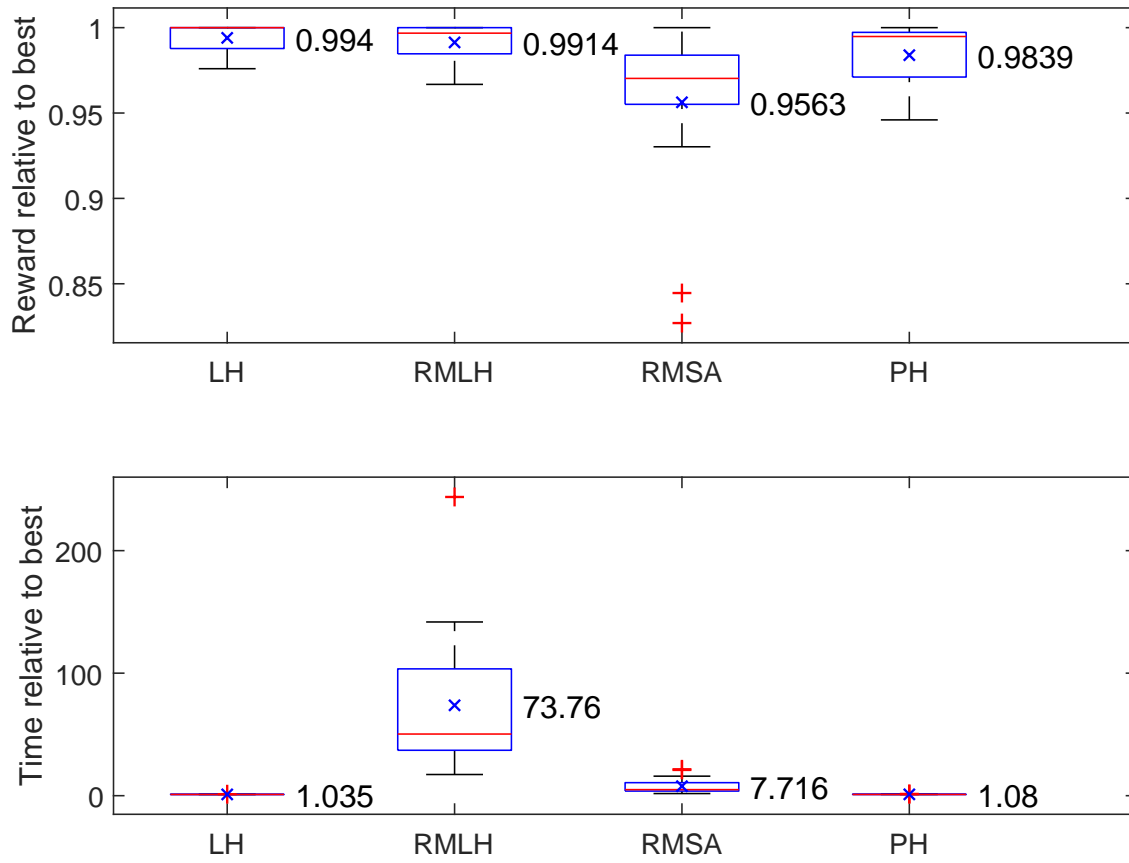


Figure 14: Visual comparison of rewards and solutions times of four algorithms relative to the maximum reward and minimum solution time respectively for each instance in Table 10.