# Machine Learning based Simulation Optimisation for Urban Routing Problems

Christopher Bayliss
Management School, University of Liverpool, Liverpool L69 7ZH, UK

March 12, 2021

**Abstract**

Many real world routing problems, including those in tourism and surveillance, can be formulated as team orienteering problems. The goal in such problems is to maximise the rewards collected by a fleet of vehicles whose routes must be completed within a time limit. This work considers a team orienteering problem set within a traffic simulation. In the stochastic environment of a road network, travel times depend on network structure, the demands of road users, driver behaviour and the congestion that arises from these. As a result travel times are difficult to predict. In this work a learnheuristic solution approach is proposed. Learnheuristics integrate machine learning and optimisation for solving combinatorial problems with inherent parameter learning problems—in this case travel times. The machine learning component is used to predict travel times based on data obtained from a limited budget of traffic simulation runs, a budget that is used within the run-time learnheuristic algorithm. In each iteration of the learnheuristic, the optimisation component utilises the travel time predictions of the machine learning algorithm to rapidly generate candidate solutions. The strongest candidate is tested in the traffic simulator, and the results of which are used to train the machine learning component. In a range of test instances, the effectiveness of different combinations of machine learning and optimisation components are tested. Experiments reveal that different combinations of machine learning and optimisation components produce solutions with different characteristics in terms of total reward and reliability. Local search followed by biased randomisation combined with a neural network was found to be effective in multiple instances. The question of how best to use the run-time of a learnheuristic is also addressed.

Keywords: Team orienteering problem, learnheuristic, traffic simulation, machine learning, metaheuristics.

## 1 Introduction

The orienteering problem, rooted in the sport of orienteering, is a much studied NP-hard problem [1] with many real world applications. These include the tourist trip design problem [2] and surveillance route planning [3]. The objective in a team orienteering problem (TOP) is to maximise the total reward collected by a fleet of vehicles, within a time limit. The TOP is often a very appropriate model for urban transportation problems, alongside vehicle routing problem models, since fleet sizes are typically constrained and route times constrained by shift duration. This work considers a TOP set within a traffic simulation. In the stochastic environment of a road network, travel times depend on network structure, the demands of road users, driver behaviour and the congestion that arises from these. As a result, travel times are difficult to predict in addition to being stochastic.

Given a traffic simulator, initially we have no information regarding the time required for travelling between consecutive targets. All we have is a time limit in which to both learn to predict inter-target travel times, by testing solutions in the simulation, and to optimise the routes of the fleet of vehicles. This work proposes a learnheuristic solution methodology to circumvent this learning and optimisation problem, which integrates heuristic routing algorithms, simulation and machine learning. The proposed learnheuristic is an iterative simulation optimisation

solution approach. In each iteration of a learnheuristic, candidate solutions are generated using a routing heuristic that uses inter-target travel time predictions that are provided by a machine learning algorithm. The best candidate solution from each iteration is tested in a simulation model, in order to reveal both the true objective value of that solution, as well as the true inter-target travel times. The true inter-target travel times are immediately used to train the machine learning algorithm, so that in subsequent iterations of the learnheuristic algorithm, the machine learning algorithm can generate more accurate inter-target travel time predictions. This simultaneous learning and optimisation approach gives rise to a free search diversification mechanism, since the machine learning algorithm predictions change over the course of the algorithm, having the effect of changing the solutions generated by the optimisation module. Figure 1 provides a flowchart of the proposed learnheuristic.

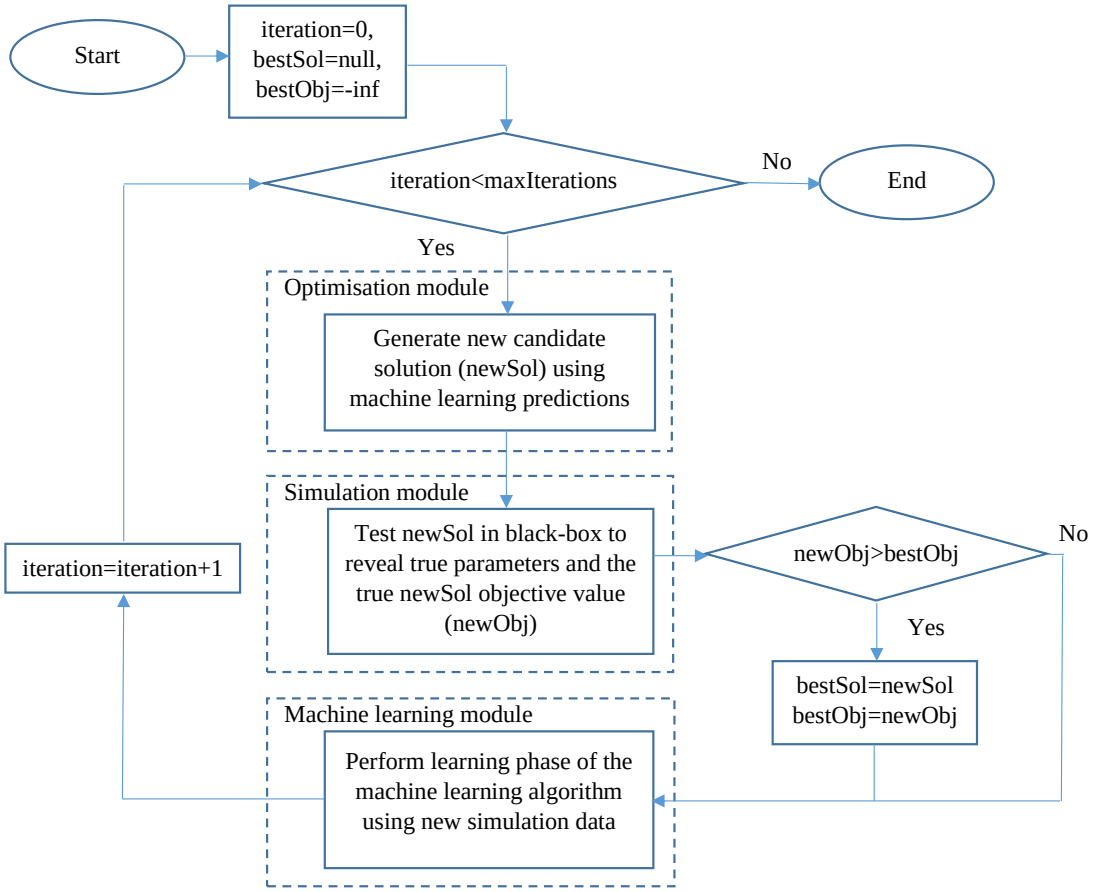


Figure 1: Overview of the proposed learnheuristic algorithm.

In order to explain how a learnheuristic is used to solve a TOP, Figure 2 provides a simple example showing how optimisation, simulation and machine learning modules are integrated within each iteration of a learnheuristic. Figure 2 (top) shows that, firstly, a solution is generated using a heuristic (optimisation module). In this example, vehicle tours are generated by adding unvisited nodes to the ends of vehicle tours, selecting the node which maximises reward divided by travel time to that node. In Figure 2 (top), the vehicle was previously assigned to visit node 1, at this point the heuristic assigns node 3 as the next node to visit, because it has the highest score. In order to evaluate next-nodes for vehicles, the heuristic uses travel time predictions provided by the machine learning module, in this example the average travel times for each road are used as the machine learning module predictions. Once a solution has been generated it is tested in a traffic simulation model (simulation module), as depicted in Figure 2 (bottom). The observed travel times from the simulation are used as training data for the machine learning module. In this example, the average travel times are updated using the observed travel times,

as depicted in the table of Figure 2 (bottom). As more iterations are performed the predictions of the machine learning module become more accurate, allowing the optimisation module to generate better solutions.

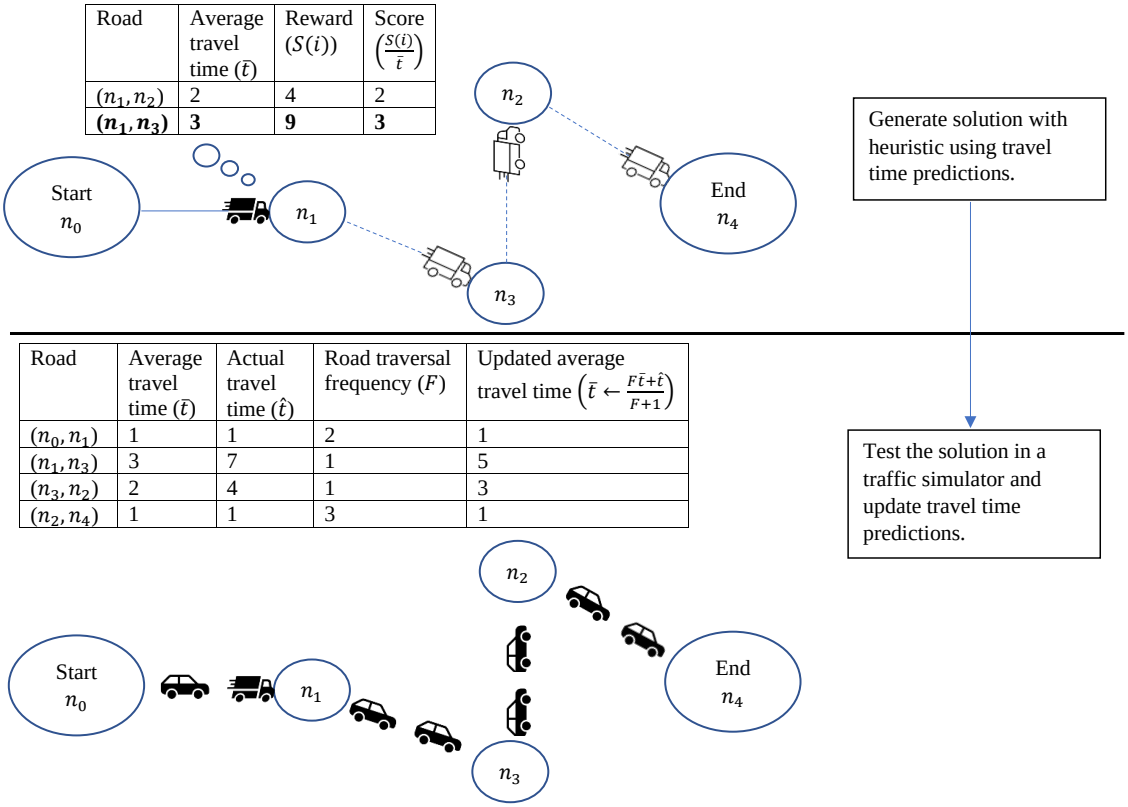


Figure 2: Basics of the proposed approach for solving the TOP using a learnheuristic.

The contributions of this work include the following. 1) The vast majority of vehicle routing problems actually take place on real road networks for which traffic is a real issue. The proposed learnheuristic approaches allows us to tackle the real problem directly in a way that is computationally tractable. 2) Previous solution methods, for deterministic and stochastic TOPs, assume prior information regarding inter-target travel times. This work aims to tackle an instance of a TOP for which no prior information is about inter-target travel times, and by doing so, proving the learnheuristic solution concept. 3) Learnheuristics are composed of three main components, a heuristic, a machine learning algorithm and a simulation model. In Section 8 the effectiveness of learnheuristics composed of different combinations of heuristics and machine learning algorithms are compared, in previous works only single combinations are considered. 4) In Section 8.3 this work addresses the question of how best to utilise the run time of a learnheuristic algorithm. Using few simulation repeats to test the best candidate solution, from each learnheuristic iteration, allows for the testing of many candidate solutions, thereby increasing the chance of finding a good solution. However, the resulting final best solution may turn out to be unreliable, due to it being overfitted to a few traffic scenarios. 5) Finally, since real road networks exhibit a number of common characteristics, such as rush hours that depend on both time of day and road direction, we gain insights into the structures of good solutions.

The structure of the remainder of this paper is as follows: Section 2 provides an overview of related literature. Section 3 provides a formal mathematical formulation of the problem under consideration. Section 4 fully defines the traffic simulation model of the learnheuristic approach. Section 5 describes the machine learning algorithms that are considered for playing the role of the machine learning module of the learnheuristic approach. Section 6 described the event based routing algorithm, upon which the considered optimisation modules are built. Section 7 defines all of the instances of a learnheuristic algorithm, namely the combinations of optimisation and machine

learning modules, that are tested in Section 8. Section 9 concludes this work by summarising the main findings and insights.

# 2 Literature review

## 2.1 Single-Vehicle and Team Orienteering Problems

The single-vehicle *orienteering problem* (OP), also known as the selective travelling salesperson problem [4], was introduced by [1]. In its simplest form the goal of an OP is to select a tour for a single vehicle which maximises the rewards collected within a specified time limit [5]. Many real world problems can be modelled as OPs, including surveillance route planning [3] and the tourist trip design problem [6]. [7] consider a time-dependent OP in which arc travel times depend on the time of day and type of arc, a set-up designed for modelling congestion effects in a road network. While assuming that travel times are known but dynamic, they solve small instances exactly and larger instances with a hybrid ant colony-local search approach. In contrast, this work solves a TOP set within a stochastic traffic simulation model and employs machine learning methods to learn to predict time-dependent travel times during the runtime of the algorithm, while assuming no prior knowledge regarding travel times.

Our work focuses on the TOP (first introduced by [8]), which extends the OP by replacing the single vehicle with a fleet of vehicles and the goal is to maximise the total rewards collected by all vehicles. Several variants of this problem are the capacitated TOP [9], TOP with time windows [10], team orienteering arc routing problem [11] and the multi-modal TOP with time windows [12]. Exact solutions have been obtained for mid-sized problems (up to 100 vertices) using column-generation [13], branch-and-price [14], cutting planes [15] and branch-and-cut [16]. Most solution approaches are still heuristic since they can be used to solve larger instances with richer constraint sets in a short amount of time. [17] consider a realistic tourist trip design problem that accounts for weather conditions, attraction opening times, congestion and personal preferences. They propose a fast and close to optimal hybrid iterated local search (ILS) and greedy randomised search procedure (GRASP) solution approach. [18] demonstrate that fast and close to optimal solutions can be obtained using a GRASP approach with path re-linking. Path re-linking involves exploring paths of solutions which join two previously generated solutions. Other proposed metaheuristic approaches include particle swarm optimisation (PSO) [16], simulated annealing (SA) [19], Pareto-mimic [20], Hybrid harmony search [21] and evolutionary algorithms (EA) [22, 23, 24]. Recent work on the OP and the TOP considers uncertainty in the rewards or the travel times [25, 26].

[3] considered a deterministic TOP involving aerial drones, while accounting for the equations of motion, a feature which brought about solution dependent decision costs. The learnheuristic methodology employed an instance-based [27] interpolation machine learning module to reduce the computational burden involved with solving the equations of motion. The results showed that the learnheuristic approach outperformed an equivalent a priori learning algorithm and provided results comparable to an equivalent reality-module-only metaheuristic but in a fraction of the time. This work builds on the methodology of [3] in numerous ways, while considering a TOP set within a stochastic traffic simulation model. Firstly, this work focusses on investigating the effect of using different combinations of machine learning and optimisation modules, whereas [3] focussed on single instances of both modules. Secondly, this work extends the optimisation module of [3] by introducing an initial solution learning mechanism, which improves solution quality in deterministic and stochastic TOP instances. Thirdly, this work addresses the issue of computational budget allocation, which is a critical issue effecting solution quality and reliability in the stochastic routing problem considered. Testing solutions using more simulation runs reduces the number of candidate solutions that can be generated within a time limit.

## 2.2 Hybrid metaheuristic solution techniques

The proposed learnheuristic presented in this work also provides a contribution to the general body of literature concerned with hybridising distinct areas of computer science with metaheuristics. In this context a related concept is that of simheuristics. Simheuristics integrate metaheuristics and simulation with the aim of generating robust solutions for complex stochastic combinatorial optimisation problems. [28] provide a review of simheuristics. Similarly matheuristics hybridise mathematical programming and metaheuriscs with the aim of achieving an algorithm with the best of both worlds, [29] provides a survey of matheuristics applied to routing problems. Within the evolutionary computing community machine learning has been extensively used to address each aspect of an evolutionary algorithm. The survey of [30] provides a comprehensive review of those presented in terms of which phase of an evolutionary algorithm machine learning has been applied. [31] introduces a taxonomy for the hybridisation of metaheuristics with other metaheuristics and metaheuristics with mathematical programming, machine learning and constraint programming. [32] enumerate opportunities for hybridising data mining techniques with metaheuristics.

Learnheuristics [33] integrate machine learning and metaheuristics with the aim solving the difficult class of combinatorial optimisation problems which feature inherent parameter learning problems.

## 2.3 Urban routing problems

Urban routing problems are characterised by the need to account for many real world details and constraints. Simulation modelling is a first resort method for modelling real world complexities. This work proposes a simulation based optimisation approach for solving urban routing problems which employs machine learning to circumnavigate the problem of large solutions times, that arise when integrating detailed simulation models within optimisation algorithms. The remainder of this section examines existing urban routing problems from the literature that may benefit from a learnheuristic solution approach. The review article of [34] considers existing research in the area of sustainable urban routing, while emphasising economic, environmental and social dimensions. The review highlights the wide array of specific urban routing problems that exist, including waste collection, electric vehicle routing and inventory routing. [35] consider the urban transit routing problem (UTRP) and propose a particle swarm optimisation algorithm. In the UTRP, the aim is to design a public transport network and the routes followed by vehicles. The aim is to minimise average transit times over a set of transportation demands, while adhering to vehicle range and total trips constraints. [36] consider the school bus routing problem (SBRP) for a Tunisian case study and propose an ant colony optimisation algorithm. [37] consider the periodic urban waste collection problem with uncertain demands, they present an interval robust mathematical formulation and an efficient simulated annealing algorithm. In experiments they demonstrate the scalability of the simulated annealing algorithm and analyse the impact of the level of uncertainty on solution costs. [38] develop a neuro-fuzzy model for evaluating arc performance costs, in a transportation network, in terms of cost, harmful emissions and noise. Training the neuro-fuzzy model on real data allows it to be used to infer such characteristics of unmeasured networks. The derived arc performance costs are used as the inputs of an adapted Clarke Wright savings routing algorithm [39] for determining delivery routes within a logistics network. [40] consider the Steiner pollution routing problem (SPRP), which accounts for time and load dependent travel time and fuel usage. A path elimination procedure is introduced which identifies emission minimising paths between delivery nodes. [40] employ closed form equations for estimating edge costs, in this work, simulation and machine learning are combined to perform the analogous task. Simulation has the advantage that it can model arbitrarily complex scenarios.

# 3 Problem Formulation

The TOP can be defined on a graph $G(N,E)$, where $N$ is a set of nodes and $E$ a set of vertices. In this work the graph $G$ is generally not a complete graph since it will represent a road network. In this work, all vehicles follow the shortest paths between pairs of nodes assigned to those vehicles. Djikstras Algorithm is used to calculate the shortest paths between each pair of nodes $(n,m) \in N^2$. The routes of vehicles are represented as a sequence of nodes assigned to those vehicles. In particular, $Y_{bk}$ denotes the $k^{th}$ node in vehicle $b$'s route, and $Y_b$ denotes the sequence of nodes assigned to vehicle $b$ in route order. This solution representation is convenient since the routing heuristics proposed in this work construct routes in time order using an event based routing algorithm (Section 6.1). The TOP set within a traffic simulation can be formulated as follows.

$$\max \sum_{\omega \in \Omega} \sum_{b \in B} \sum_{k \in \{1..|Y_b|\}} g_{\omega b} S(Y_{bk}). \tag{1}$$

Objective (1) is that of maximising the total expected reward collected by the fleet of vehicles $B$. $g_{\omega b}$ is a binary variable that takes a value of 1 if vehicle $b$ completes their assigned route $Y_b$ within the time limit $T_{max}$ in simulation test $\omega$ out of a set $\Omega$ of simulation tests. The function $S(n)$ returns the reward associated with a visit to a node $n \in N$.

$$g_{\omega b} T_{\omega}(Y_b) \leq T_{max}, \ \forall b \in B, \ \forall \omega \in \Omega. \tag{2}$$

Constraint (2) represents the time limit on each vehicles' route. The function $T_{\omega}(Y_b)$ returns the time required for vehicle $b$ to complete their assigned route $Y_b$ in simulation scenario $\omega$. In this work the role of the function $T_{\omega}(Y_b)$ is played by the traffic simulation presented in Section 4. If vehicle $b$ fails to complete their assigned route within a total time of $T_{max}$, then Constraint (2) is satisfied only if $g_{\omega b} = 0$, in which case, all of the rewards collected by vehicle $b$ are lost, as expressed by Objective (1). If vehicle $b$ completes their assigned route within a total time of $T_{max}$ then $g_b = 1$ since this will help to maximise Objective (1).

$$\sum_{b \in B} \sum_{k \in \{1..|Y_b|\}} M(Y_{bk}, n) \leq 1, \ \forall n \in N. \tag{3}$$

Constraint (3) states that each node can only be assigned to one vehicle's route. This constraint ensures that at most one reward can be collected from each node. The function $M(Y_{bk}, n)$ returns 1 if node $n$ is the $k^{th}$ node in vehicle $b$'s route. It is worth noting that in the context of a non-complete graph, each node can be visited more than once, however the reward collection activity of each node can only be assigned to one vehicle. Other possible visits will correspond to vehicles following the shortest path between a pair of nodes assigned to their routes'.

$$Y_{b0} = n_0, \ \forall b \in B. \tag{4}$$

Constraint (4) specifies that the first node in each vehicle's route is the start depot $n_0$.

$$Y_{b|Y_b|} = n_{|N|}, \ \forall b \in B. \tag{5}$$

Constraint (5) specifies that the last node in each vehicle's route is the end depot $n_{|N|}$.

$$Y_{bk} \in N, \ \forall b \in B, \ \forall k \in \{1..|Y_b|\}. \tag{6}$$

Constraint (6) specifies that all nodes in each vehicle's route belongs to the set of nodes $N$.

The following sections introduce the simulation model (Section 4), machine learning (Section 5) and optimisation (Section 6) modules of the proposed learnheuristic.

# 4 Traffic Simulator

The traffic simulation model introduced in this section is a discrete-time continuous-space single-lane car-following-model (DTCSSLCFM), which accounts for driver reaction time and anticipatory driver behaviour. Car-following-models [41] are based on modelling driver behaviour that is governed by the competing aims of travelling to a desired destination as quickly as possible and avoiding crashing into the vehicle in front. DTCSSLCFM is also very similar to cellular automata approaches for modelling single-lane traffic flow, such as in [42], except DTCSSLCFM accounts for continuous space, driver reaction time and anticipatory driver behaviour, in a way similar to that considered in [43]. There is a vast and rich literature relating to traffic flow modelling, analysis and forecasting. A large amount of these employ fluid flow models of traffic flow. In particular, [44] propose a fluid flow model which accounts for driver reaction time and relaxation time, the latter captures the time before drivers adjust their target velocity to that of the local prevailing traffic conditions. [45] provides a detailed survey of microscopic traffic flow models, which are based on modelling traffic at the vehicle level. Their work includes empirical data consistent with car-following-models in which driver behaviour is highly dependent upon the distance to the car in front. In this work the DTCSSLCFM provides an initially unknown and stochastic environment in which we are tasked with solving a TOP.

The state of the overall traffic simulation is updated according to Algorithm 1. Algorithm 1 is used to simulate traffic flow in a road network consisting of a set of connected single-lane roads. Algorithm 1 simulates traffic flow in a road network by repeatedly simulating single time intervals of traffic flow on each road in the network until the simulation clock time exceeds the maximum time limit $T_{max}$. Algorithm 1 features an optional run in period, in which traffic flow is simulated without beginning the tours of the vehicles in the TOP fleet $B$ (line 4).

In each time interval of Algorithm 1 traffic demand distributions (Section 8) are sampled and any new routes are initialised by adding the associated vehicle to the queue for entering the first road on their assigned route (line 12). The role of the queues for entering roads is two-fold: 1) this approach removes the need for traffic lights, such as that considered by [46], and instead reflects a case where there exists a unique exit junction for each possible next road; 2) this approach prevents the occurrence of gridlocks since the next road queues remove vehicles from the traffic simulation whilst they cannot enter their next road, allowing vehicles behind them to continue along their assigned routes. When a vehicle enters an empty queue for their next road, their velocity remains the same, otherwise it is set to 0. As a result, traffic flow can be disrupted for two reasons, driver behaviour and queues for entering the next road on a vehicle's tour.

All vehicles, except for those in the fleet $B$, follow the shortest path between the first node and end node of their tour. An alternative model might consider user equilibrium routes [47] in which all vehicles individually attempt to learn how to minimise their journey times.

**Algorithm 1** Traffic simulation and TOP tour reward evaluation.

Inputs: *runIn* (length of run in period), $T_{max}$ (time limit), $\Delta t$ (time interval length), $Y$ (vehicle tours)
$t \leftarrow -runIn$ (initialise time to minus run-in period), $H \leftarrow \emptyset$ (initialise the set of successfully completed TOP tours)
**while** $t < T_{max}$ **do**
  **if** $t \geq 0$ (begin TOP tours) **then**
    **for** $b \in B$ **do**
      //Add vehicle $b$ to the queue for entering the first road on their tour
      $addToQueue\left(r_{y_{b0},y_{b1}}, b\right)$
    **end for**
  **end if**
  **for** $r \in R$ **do**
    $oneIterationOfRoad\,(r)$ (Algorithm 2)
  **end for**
  //Sample traffic demand distributions and add generated random routes to the queues for the first roads on their assigned routes
  $t \leftarrow t + \Delta t$
**end while**
//Calculate the total reward collected by TOP vehicles that completed their tours within the time limit
$reward = 0$
**for** $b \in H$ **do**
  **for** $n \in Y_b$ **do**
    $reward \leftarrow reward + S(n)$
  **end for**
**end for**
**Output:** *reward*

Within each time interval considered in Algorithm 1 a time interval of traffic flow is simulated on each road using Algorithm 2. Algorithm 2 firstly considers whether or not there is enough space at the start of the road for the vehicle at the front of the queue for entering that road. If there is, that queued vehicle is added to the road, otherwise the velocity of the queued vehicle is set to zero as to simulate congestion at the junction of the road. Following this Algorithm 2 considers, in turn, each vehicle on the road in order of increasing arrival time on the current road (i.e., starting with the vehicle at the front) and applies driver behaviour Algorithm 3 to simulate one time interval ($\Delta t$) of driver behaviour and vehicle motion.

**Algorithm 2** $oneIterationOfRoad\,(r)$.

Inputs: $r$ (current road), $\Delta t$ (time interval length), $I$ (list of vehicles on the current road in increasing order of arrival time), $x$ (vector of vehicle positions on road $r$ in increasing order of arrival time on road $r$), $v$ (vector of vehicle velocities in increasing arrival time order), $q$ (vector of queued vehicles waiting to begin traversing road $r$ in increasing order of queue arrival time)
//Can the vehicle at the front of the queue enter the current road
**if** $x_{|x|} > length(q_1)$ **then**
  //This vehicle can enter this road with the same velocity it had at the end of its previous road
  $x_{|x|+1} \leftarrow 0$
  $v_{|x|+1} \leftarrow velocity(q_1)$
  $I \leftarrow I \cup \{q_1\}$
**else**
  //This vehicle could not enter this road with the same velocity it had at the end of its previous road
  $velocity(q_1) \leftarrow 0$
**end if**
//Consider each vehicle on this road in decreasing order of arrival time on this road
**for** $i \in I$ **do**
  //Apply driver behaviour algorithm to vehicle $i$ (Algorithm 3)
  $driverBehaviour\,(i)$
**end for**

Each vehicle's behaviour in a single time intervals is governed by Algorithm 3. Algorithm 3 is designed to simulate the behaviour of a single vehicle in a single time interval in single-lane traffic. Algorithm 3 assumes that vehicle $i$ has a constant braking rate (deceleration) $D_i$ and an acceleration that decreases, from the maximum $A_i$, linearly with increasing velocity up to the vehicle's maximum velocity $VMax_i$. The acceleration of vehicle $i$ in one time interval depends on the relative position and velocity of the vehicle $(i-1)$ in front of vehicle $i$. If vehicle $i$ is catching up with vehicle $i-1$ (line 5) then it may be the case that vehicle $i$ cannot slow down enough to avoid crashing with vehicle $i-1$. The relative velocity and braking rate of vehicle $i$ determines the distance ($d_{min}$) required by vehicle $i$ to slow down to the same velocity as vehicle $i-1$ (line 7), if $d_{min}$ is greater than or equal to the distance $d$ between vehicles $i$ and $i-1$ then vehicle $i$ must decelerate in order to avoid crashing (line 10), otherwise vehicle $i$ may accelerate (line 13). The equations of lines 7 and 10 are derived from the constant acceleration kinematic equation $v_f^2 = v_0^2 + 2D_i d_{min}$, where $v_f = 0$ is the final desired relative velocity, $v_0 = v_{i-1} - v_i$ is the initial relative velocity. If vehicle $i$ is not catching up with vehicle $i-1$ (line 16) then vehicle $i$ will try to accelerate (line 22) unless the distance to vehicle $i-1$ ($d$) is less than the desired gap $\delta$ to vehicle $i-1$, in which case the brakes are applied. When a vehicle is not catching up with the vehicle in front, the desired gap to the vehicle in front $\delta$ is calculated (line 3) as the driver's thinking time multiplied by their current velocity plus the length of their vehicle, this gaps ensures that, if the vehicle in front brakes, the vehicle will have time to react and adjust their speed accordingly. Once the desired or necessary acceleration (or deceleration) rate of vehicle $i$ has been determined then the position $x_i$ and velocity $v_i$ of vehicle $i$ is updated (lines 26 and 27). Vehicle $i$ is not allowed to reverse or crash with the vehicle in front. If vehicle $i$ has reached the end of its current road and there exists a next road in their assigned route $r'$, then vehicle $i$ is added to the queue for entering its next road. If a vehicle in the TOP vehicle fleet $B$ has reached the end their tour, that vehicle is added to a list of vehicles $H$ that have successfully completed their assigned tours within the time limit $T_{max}$, this list is required for determining the total reward and reliability level of a solution to the TOP. In this work $A_i = D_i = 20$, $VMax = 20$, $l_i = 10$, $\Delta\tau_i = 0.5$ for all vehicles. Note that the values for $A_i$, $D_i$ and $Vmax_i$ have the potential to significantly change the simulation results. In general lower values for $A_i$, $D_i$ and $Vmax_i$ each have the effect of reducing average speeds. Lower $D_i$ values mean vehicles have to brake earlier, which also reduces average speeds. Lower average speeds mean vehicles spend longer on the road. As a result, more vehicles will be on the road at the same time, which increases congestion, which in turn reduces average speeds further.

**Algorithm 3** $driverBehaviour(i)$.

Inputs: $\Delta\tau_i$ (Driver $i$ thinking and reaction time), $v_i$ and $v_{i-1}$ (current vehicle velocity and that of the vehicle in front), $x_i$ and $x_{i-1}$ (current vehicle position and that of the vehicle in front), $D_i$ (maximum deceleration of vehicle $i$), $l_i$ (length of vehicle $i$), $A_i$ (maximum acceleration of vehicle $i$), $VMax_i$ (maximum velocity of vehicle $i$), $\Delta t$ (time interval size), $r$ (current road), $r'$ (next road)
$d = x_{i-1} - x_i$ (distance between the rear of vehicle $i$ and rear of vehicle $i-1$)
$\delta = \Delta\tau_i v_i + l_i$ (vehicle $i$'s desired minimum gap to vehicle $i-1$)
$v_{rel} = v_{i-1} - v_i$ (relative velocity of vehicle $i-1$ compared to that of vehicle $i$)
**if** $v_{rel} < 0$ **then**
    //Vehicle $i$ is catching up with vehicle $i-1$
    $d_{min} = \frac{v_{rel}^2}{2D_i}$ (minimum distance in which vehicle $i$ can decelerate to the same velocity as vehicle $i-1$)
    **if** $d \leq d_{min}$ **then**
        //Collision course, action required
        $a = -\frac{v_{rel}^2}{2(d-d_{min})}$ (calculate vehicle $i$'s required deceleration including any required emergency braking)
    **else**
        //No collision risk, accelerate
        $a = A_i\left(1 - \frac{v_i}{VMax_i}\right)$ (vehicle $i$'s acceleration)
    **end if**
**else**
    //Vehicle $i$ is not catching up with vehicle $i-1$
    **if** $d \leq \delta$ **then**
        //Vehicle $i$ is too close to vehicle $i-1$, apply the brakes
        $a = -D_i$ (deceleration is assumed to be independent of the vehicle's velocity)
    **else**
        //No collision risk, accelerate
        $a = A_i\left(1 - \frac{v}{VMax_i}\right)$;
    **end if**
**end if**
//Update vehicle $i$'s position and velocity
$x_i \leftarrow min\left(x_{i-1} - \delta, x_i + (\Delta t v_i) + \left(0.5a\left(\Delta t^2\right)\right)\right)$ (no crashing)
$v_i \leftarrow min\left(VMax_i, max\left(0, v_i + \Delta t a\right)\right)$ (no reversing or exceeding the vehicle's maximum velocity allowed)
//Has vehicle $i$ reached the end of its current road
**if** $x_i + l_i > length(r)$ **then**
    **if** $\exists r'$ **then**
        //Add vehicle $i$ to the queue for entering the next road on their route
        $addToQueue(r', i)$
    **else**
        **if** $i \in B$ **then**
            //Update the list of TOP tours successfully completed
            $H \leftarrow i$
        **end if**
    **end if**
**end if**

## 4.1 Validation of traffic simulation

This section shows that the proposed traffic simulation model is consistent with known results of traffic simulators. Firstly, the case of a single circular road is considered and known results from the literature regarding the effect of increasing the traffic density are recreated.

Figure 3 shows that the traffic model recreates the well known structure of the fundamental diagram of traffic flow [42, 44] relating flow rate and vehicle density. Additionally, it shows the effect of driver reaction time on flow rate and average velocity dependent upon the number of vehicles. Figure 3 (left) shows that flow rate increases linearly and independently of driver reaction time for up to around 30 vehicles. Accordingly, Figure 3 (right) shows that average vehicle velocities remain constant up until around 30 vehicles. As the number of vehicles

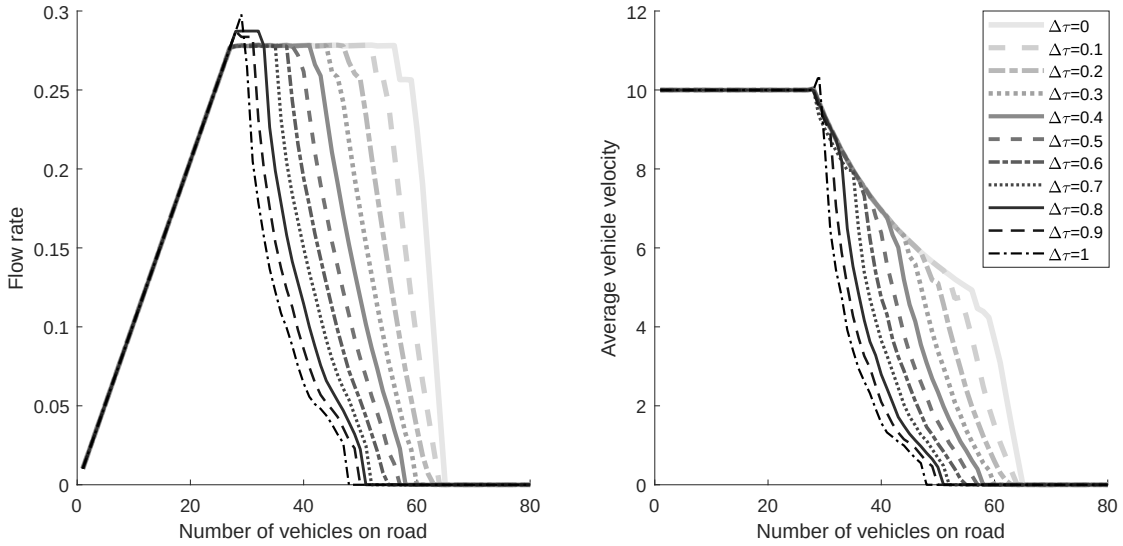


Figure 3: Flow rate and velocity curves by reaction time.

on the circular road increases above 30 congestion increases and the average vehicle velocity decreases. Figure 3 (left) shows that an optimal flow rate is maintained despite the presence of some congestion (more vehicles going slower) and that the optimal flow rate is maintained for higher vehicle densities for quicker driver reaction times. This is because quicker reaction times decrease the minimum safe distance between vehicles, effectively increasing the road's capacity.

Figure 4 provides a space-time plot of vehicle positions on the circular road for several key vehicle densities which can be matched with different regions of Figure 3. The angle of the black arrow in each plot relative to the time axis indicates vehicle velocity, which remains constant until congestion occurs, after which vehicle velocity decreases until a grid lock situation occurs. For the case of the bottom-left and bottom middle panes of Figure 4, the widely known *stop-go waves* [48] develop on the circular road, visible as two-different vehicle velocities dominating two different sections of the road.

Figure 5 provides a plot of road traversal times after different times for example roads and from several different runs of the simulation. The two roads correspond to a bottleneck in the example road network considered. Figure 5 shows that road traversal times are strongly dependent upon road arrival time. Figure 5 also suggests there is a strong stochastic component to road traversal times. The work [49] illustrates empirical data which exhibit equally noisy characteristics for both road traversal times at different times and for different days.

# 5 Machine Learning Modules

In this work machine learning algorithms are used to provide fast approximations of road traversal times at different time of the day. The problem of predicting the time ($\hat{t}$) required to traverse the road joining nodes $i$ and $j$ ($r_{ij}$) depending upon the arrival time ($\alpha$) can be represented as follows.

$$\hat{t} \leftarrow f_{ij}(\alpha) \tag{7}$$

That is, a separate instance of a machine learning algorithm is trained for each road using the data gathered from the simulation. Whenever a vehicle in $B$ reaches the end of a road $r_{ij}$ in Algorithm 3, a tuple composed of the vehicle's arrival time on that road and the road traversal time $(\alpha,\hat{t})$ is added to the set of data regarding traversals of that road ($Data_{ij} \leftarrow Data_{ij} \cup \{(\alpha,\hat{t})\}$. If new data has been collected regarding traversals of road $r_{ij}$ during an

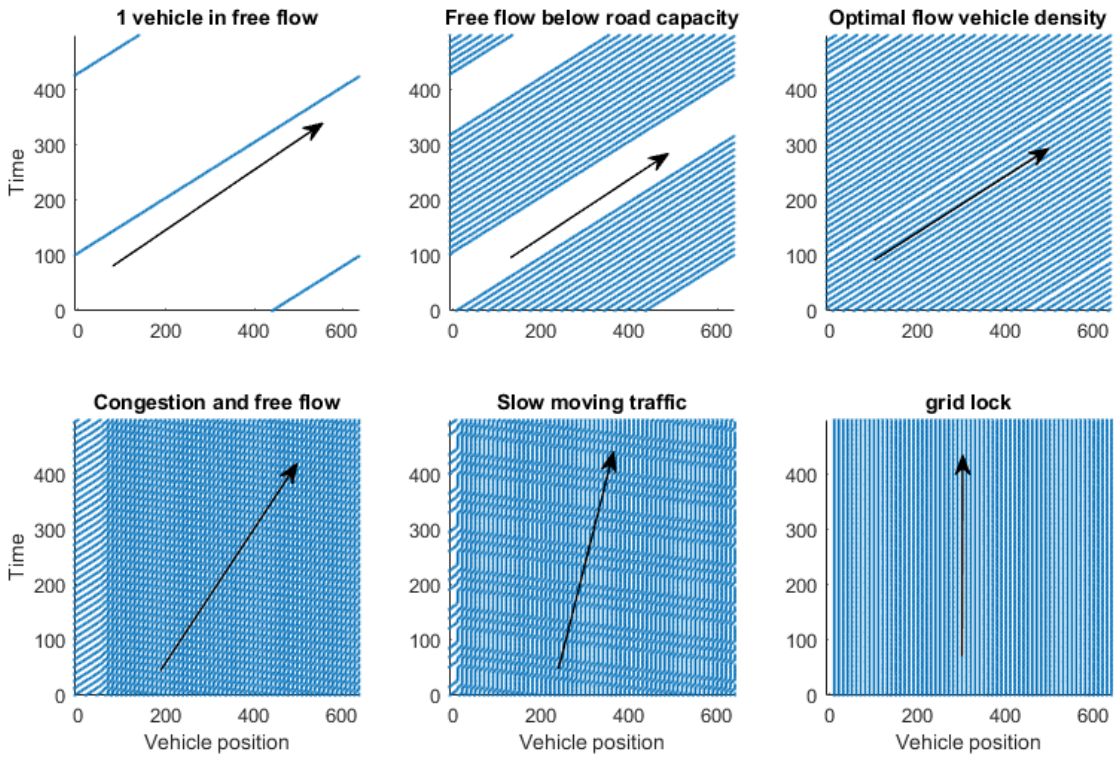


Figure 4: Space-time plot of vehicle positions at several key vehicle densities.

iteration of the learnheuristic algorithm, then a training iteration is performed for the machine learning algorithm devoted to road $r_{ij}$. The machine learning algorithms that will be considered in this work are listed as follows.

- *Neural network* (NN). The neural network architecture used in this work (based on [50]) consists of a feed-forward neural network composed of 1 hidden layer of 5 neurons, one input (road arrival time) and one output (predicted road traversal time). In experiments, more complex architectures offered little or no benefit in terms of overall prediction accuracy. The activation function of each neuron is *tanh*. Training is performed using the back propagation algorithm. In particular, if new data for a road is generated in an iteration of the learnheuristic, all data collected for the road in question is feed forward through the network, the prediction error is calculated. For each forward pass, the backpropagation algorithm [51] is applied to update the neuron weights. The learning rate is controlled by applying a learning rate parameter $\eta$ to the weight changes computed by the back propagation algorithm. Based on experiments, initially $\eta = 0.1$ which is decreased over the course of the learnheuristic algorithm to $\eta = 0.0001$, according to a polynomial curve with exponent 0.5. Road traversal predictions are obtained by performing a forward pass of the feed forward network, with the candidate road arrival time as the input for the input neuron. Additionally, the input and output values of the training data are each linearly mapped to values between -1 and 1 (the domain of the activation function). The output predictions are mapped back using the inverse mapping of the range of the prediction problem.

- *Piece-wise linear regression* (PWL). In order to determine the number of linear segments, the formula $pieces = \max\left(1, round\left(\frac{log|Data_{ij}|}{log(segmentsBase}\right)\right)$ is used, where *segmentsBase* is a parameter which controls the number of linear segments through the relation $|Data_{ij}| = segmentBase^{pieces}$, so that if $segmentsBase = 2$ and $|Data_{ij}| = 32$, then $pieces = 5$. Based on experiments $segmentsBase = 40$ was found to work well, which corresponds to 3 or 4 segments for roads frequently traversed during a run of the learnheuristic algorithm. The training step for this machine learning algorithm applies ordinary least squares linear regression (OLS) to each of *pieces* segments of data, where the data is sorted in order of increasing road arrival time.

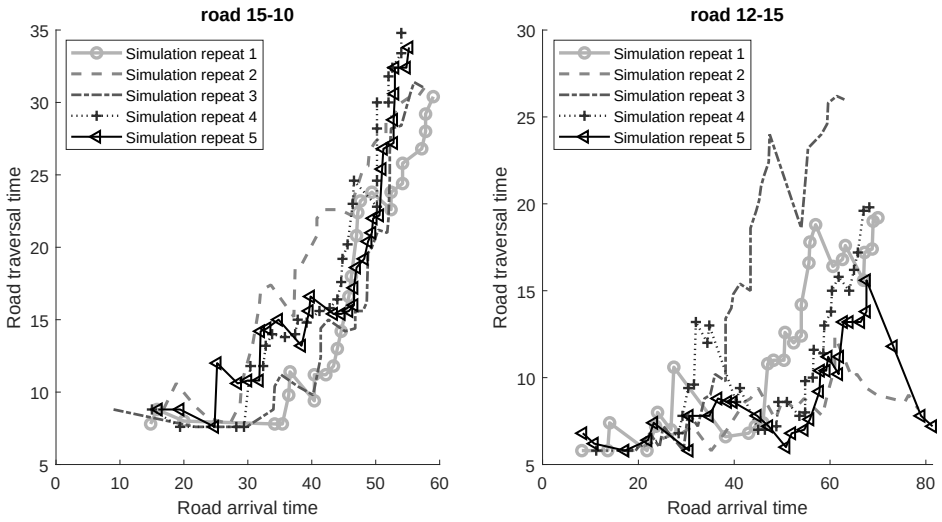


Figure 5: Road traversal times at different times of the day, in different repeat runs of the simulation model.

Each segment receives an equal share of the available data. Road traversal time predictions are generated by identifying the linear segment relevant to the candidate road arrival time and applying the corresponding linear equation.

- *Convex hull* (CH). This machine learning algorithm is based on finding the set of linear segments which define the convex hull of the traversal time data for each road (see second row first column of Figure 6). Predictions are generated in an analogous way to that of the piece-wise linear regression.

- *Instance based average* (IBAve). This machine learning algorithm is based identifying the data point in $Data_{ij}$ with the closest road arrival time and returning the road traversal time of that data point. If there are numerous data points with the same arrival time, the average road traversal time of those data points is returned as the predicted road traversal time. [49] suggested a nearest neighbour travel time prediction approach based on finding the previous day with the most similar vehicle velocities up until the current time.

- *Instance based minimum* (IBMin). This machine learning algorithm is the same as *Instance based average* except that ties are broken by selecting the minimum road traversal time of the data points with nearest (and equal) road arrival time. The purpose behind the inclusion of this machine learning algorithm is that breaking ties using the minimum road traversal times may provide optimistic traversal time prediction, at the risk of a low reliability rate.

- *Instance based maximum* (IBMax). This machine learning algorithm is the same as *Instance based average* except that ties are broken by selecting the maximum road traversal time of the data points with nearest (and equal) road arrival time.

- *Overall average* (OAve). This machine learning algorithm returns the average road traversal time over all of the data stored for a road as the predicted road traversal time.

- *Overall minimum* (OMin). This machine learning algorithm returns the minimum road traversal time over all of the data stored for a road as the predicted road traversal time.

- *Overall maximum* (OMax). This machine learning algorithm returns the maximum road traversal time, over all of the data stored for a road, as the predicted road traversal time.

- *No learning* (NL). This machine learning algorithm assumes that there is no traffic and each vehicle is free to travel at their maximum velocity. This approach is used as the default prediction method in cases where a machine learning algorithm does not have enough data to generate a prediction.

Only NN and PWL have parameters that need to be tuned, the other approaches are parameterless. The reported parameters for NN and PWL are based on experiments. For NN, a wide range architectures lead to very similar, and good, performance.

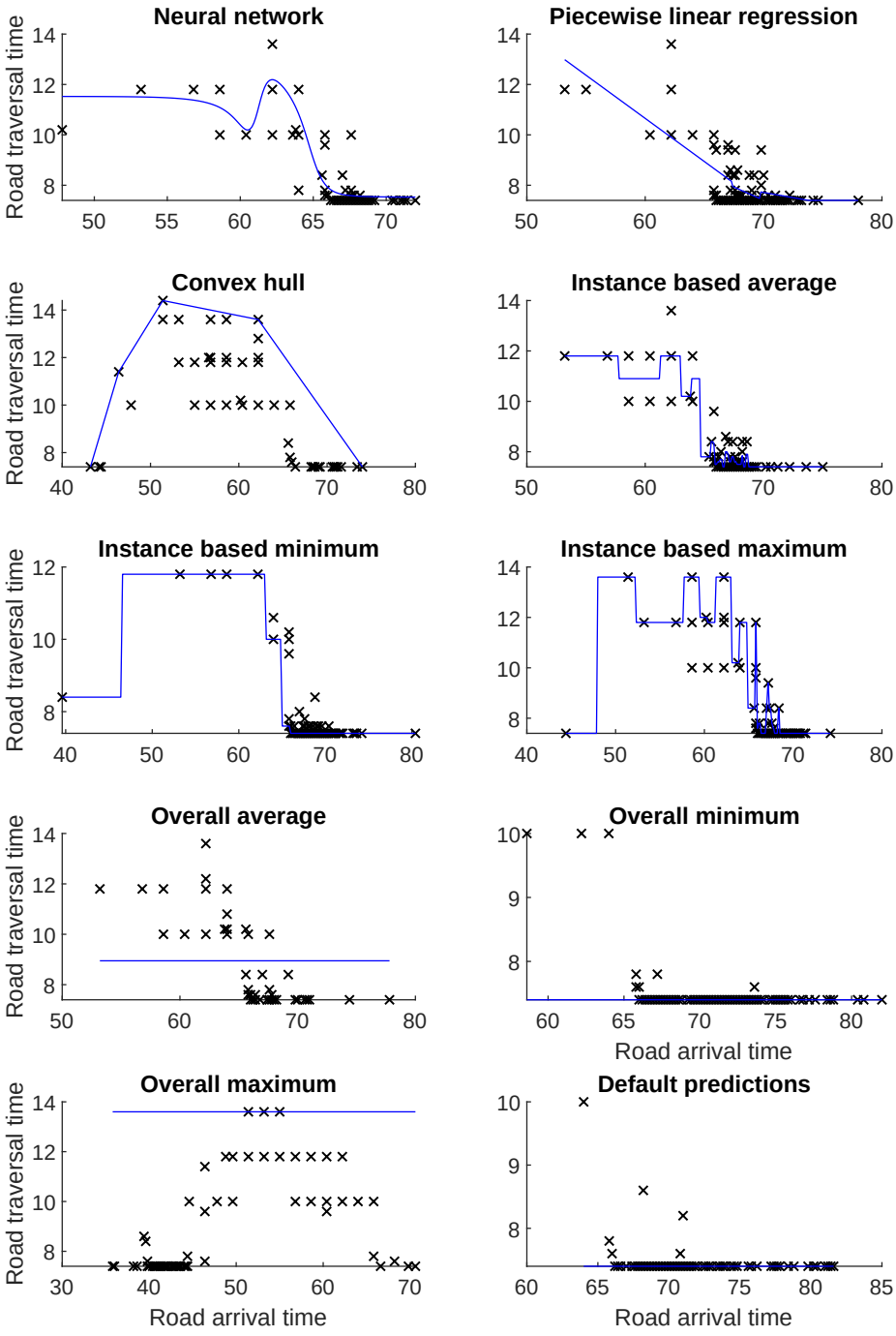


Figure 6: Visual comparison of predicted road traversal times, for each machine learning module (solid lines), and the data (crosses), collected during the run time of the learnheuristic, used to training each machine learning module.

Figure 6 illustrates examples of predicted road traversal times, at different road arrival times, for each machine learning algorithm described above. Figure 6 illustrates the difficulty of the prediction problem owing to the fact that equal road arrival times can lead to different road traversal times. The reason for this is that road traversal times are dependent upon the whole route plan preceding a road traversal, i.e., what was referred to as solution dependent decision costs earlier. The stochastic travel demands of other road users is also a factor.

# 6 Optimisation Modules

In this section, 5 optimisation modules are proposed, that are each based on an event based routing algorithm, which is described in Section 6.1. The particular details of the 5 optimisation modules are provided in Section 6.4. Between them they combine local search, biased randomisation and a promising solution look-up-table learning mechanism.

## 6.1 Event based routing algorithm

In this work, the optimisation modules of the proposed learnheuristic are each based on a constructive event based routing algorithm (Algorithm 4). Similar approaches have been considered in the past [52]. Event based routing algorithms build vehicle routes in a time ordered fashion, which is an efficient heuristic approach for routing problems with time dependent travel times. This approach can also easily handle many difficult constraints. In each iteration of such algorithms, the vehicle that is expected to reach their last assigned node next ($b$) is assigned a next node from the set of unassigned nodes $U$. The decision for which next node is selected for vehicle $b$ is based on the candidate node ($i$) which maximises a weighted sum of a number of efficiency attributes, $score(i) = \sum_{j \in J} w_{bj}\lambda_{ij}$. $J$ is the set of efficiency attributes, $w_{bj}$ is the weight that vehicle $b \in B$ gives to efficiency attribute $j$ and $\lambda_{ij}$ is the efficiency score of attribute $j$ for candidate node $i$. In order to reduce the issue of symmetrical weight solutions, each weight is divided by the absolute value of the weight with the largest absolute value. The efficiency attributes that are considered in this work are listed as follows.

1. *Distance to next node n.* $\lambda_{i1} = L_{l,n}$, where $L$ is the shortest path distance matrix and node $l$ is the last node assigned to vehicle $b$, $l = Y_{b,|Y_b|}$. Giving a large negative weight to this attribute simulates a nearest neighbour routing heuristic.

2. *Travel time to next node n.* $\lambda_{i2} = f_{l,n}(\alpha_b)$, see Equation (7), $\alpha_b$ denotes the predicted time at which vehicle $b$ reaches its last assigned node $l = Y_{b,|Y_b|}$. The value of this attribute is provided by the machine learning module of a learnheuristic algorithm.

3. *Average speed on the trip to the next node.* $\lambda_{i3} = \frac{\lambda_{i1}}{\lambda_{i2}}$. Giving a large positive weight to this attribute encourages routes with high average velocities, which may also help them to visit more targets within the time limit.

4. *Reward associated with the next node i.* $\lambda_{i4} = S(i)$. Giving a large positive weight to this attribute encourages routes that prioritise the visiting nodes with high reward values.

5. *Reward per unit of time on the trip to the next node.* $\lambda_{i5} = \frac{\lambda_{i4}}{\lambda_{i2}}$. Giving a large positive weight to this attribute encourages routes which collect rewards at a high rate.

6. *Reward per unit distance on the trip to the next node.* $\lambda_{i6} = \frac{\lambda_{i4}}{\lambda_{i1}}$. Giving a large positive weight to this attribute encourages routes which collect rewards while covering as little distance as possible.

7. *Distance to the end depot from candidate next node i.* $\lambda_7 = L_{i,n_{|N|}}$, where node $n_{|N|}$ is the end depot. Giving a large positive weight to this attribute encourages vehicles to stay as far away as possible from the end

depot, which can be a useful mechanism for ensuring the rewards are collected from nodes which are most distant from the end depot.

Efficiency attributes 1-7 are calculated for each non-assigned node every time Algorithm 4 considers assigning an additional node to a vehicles route. The event based constructive algorithm is given in Algorithm 4 and is adapted from that introduced previously in [3]. Algorithm 4 is summarised as follows. The inputs of Algorithm 4 consist of the efficiency attribute weight matrix $w$, geometric distribution shape parameter $\beta$, and a random seed $\sigma$. The purpose of $w$ is described above. The purpose of the geometric distribution shape parameter $\beta$ is to introduce a controlled degree of randomness into the selection of the next nodes that are assigned to vehicles. This approach is referred to as biased randomisation [53]. $\beta = 1$ results in greedy node selection with respect to $score(n)$, $\beta$ close to but lower than one reduces the level of greediness of node selection whilst still retaining the underlying logic of the greedy event based constructive algorithm and provides a mechanism for generating a range of alternative solutions by mere repeated execution of Algorithm 4. The random seed $\sigma$ allows us to control the behaviour of Algorithm 4 even when $\beta < 1$. This is useful for two reasons: i) previous routes can be regenerated using their corresponding values of $w$, $\beta$ and $\sigma$; ii) it allows us to perform a sensitivity based local search of the weight matrix $w$ by fixing the values of $\beta$ and $\sigma$ during one cycle of a multistart local search algorithm (Section 6.2). Following this, lines 2-6 of Algorithm 4 set the start depot node as the first node in each vehicle's route, reset the set of non-assigned nodes, reset the random number generator using the random seed $\sigma$, resets the next node arrival times ($nextArrivalTime(b)$) of all vehicles to 0 and generates a list of vehicles sorted according to next arrival time ($\mathfrak{B}$). Algorithm 4 then proceeds, in lines 7-36, to iteratively select next nodes for the next vehicle to reach their last assigned node (line 9). Lines 12-19 calculate attractiveness scores for each remaining non-assigned feasible next node. Lines 21-30 sort the candidate nodes according to attractiveness score (line 23), select one according to a geometric distribution (line 25), add that node to the vehicle's route (line 26), update the vehicle's next node arrival time (line 28) using the travel time predictions generated by the machine learning module and update the next node arrival time sorted list of vehicles $\mathfrak{B}$ (line 30). If Algorithm 4 fails to find a node that can be feasibly added to a vehicle's route, then that vehicle's route is finalised by assigning the end depot node as their next node (line 33), and removing that vehicle from the next node arrival time sorted list $\mathfrak{B}$ (line 34). Once no more vehicles can be assigned any more nodes, the algorithm terminates and returns the generated routes $Y$ (line 37).

**Algorithm 4** Event-based constructive vehicle-routing algorithm

1: Inputs: Efficient attribute weight matrix $w$, random seed $\sigma$, geometric distribution shape parameter $\beta$
2: Reset vehicle routes $Y_b \leftarrow n_0, \forall b \in B$
3: Reset the set of non-assigned nodes $U \leftarrow N \setminus \left\{n_0, n_{|N|}\right\}$
4: Reset the random number generator using the random seed $\sigma$
5: Set $nextArrivalTime\,(b) \leftarrow 0, \forall b \in B$
6: Let $\mathfrak{B}$ be a list of vehicles ($b \in B$) sorted in increasing next arrival time order
7: **while** $|\mathfrak{B}| > 0$ **do**
8: // Select the vehicle ($\mathfrak{b}$) expected to reach its last allocated node soonest as the vehicle that will be allocated an non-assigned node next
9: $\mathfrak{b} \leftarrow \mathfrak{B}_1$
10: Reset the list of candidate nodes ($C \leftarrow \emptyset$) that vehicle $\mathfrak{b}$ can visit next
11: // Consider the remaining non-assigned nodes as potential candidates for the next node for vehicle $\mathfrak{b}$ to visit next
12: **for** $n \in U$ **do**
13: // Check the feasibility of vehicle $\mathfrak{b}$ visiting node $n$ using ML module edge traversal time predictions
14: **if** vehicle $\mathfrak{b}$ can visit node $n$ and return to the end depot before time $T_{max}$ **then**
15: // Calculate efficiency attributes (Section 6.1) for vehicle $\mathfrak{b}$ visiting node $n$ next and the associated overall attractiveness score ($score\,(n)$)
16: $score\,(n) \leftarrow \sum_{j \in J} w_{\mathfrak{b}j} \lambda_{nj}$
17: // Add node $n$ to the candidate list $C$
18: $C \leftarrow C \cup \{n\}$
19: **end if**
20: **end for**
21: **if** $|C| > 0$ **then**
22: // Sort the list of candidate nodes in order of decreasing attractiveness score
23: $Sort\,(C, score, \text{'decreasing'})$
24: // Select a node $n_{i*}$ from $C$ according to geometric distribution with shape/greediness parameter $\beta$
25: $i* = Mod\left(\left\lfloor \frac{log(uniRand(0,1))}{log(1-\beta)} \right\rfloor, |C|\right)$
26: $Y_{\mathfrak{b}} \leftarrow (Y_{\mathfrak{b}},\ n_{i*})$
27: // Update vehicle $\mathfrak{b}$'s next arrival time according to the ML module's predicted edge traversal time to node $n_{i*}$
28: $nextArrivalTime\,(\mathfrak{b}) \leftarrow nextArrivalTime\,(\mathfrak{b}) + t'\left(Y_{\mathfrak{b}\left(|Y_{\mathfrak{b}}|-1\right)}, n_{i*}\right)$
29: // Sort $\mathfrak{B}$ in increasing order of $nextArrivalTime$
30: $Sort\,(\mathfrak{B}, nextArrivalTime, \text{'ascending'})$
31: **else**
32: // No feasible candidate nodes exist for vehicle $\mathfrak{b}$, set the end depot as the vehicles next node and remove vehicle $\mathfrak{b}$ from the vehicle list $\mathfrak{B}$
33: $Y_{\mathfrak{b}} \leftarrow \left(Y_{\mathfrak{b}},\ n_{|N|}\right)$
34: $\mathfrak{B} \leftarrow \mathfrak{B} \setminus \{\mathfrak{b}\}$
35: **end if**
36: **end while**
37: Output: vehicle routes $Y$

## 6.2 Sensitivity Based Local Search

The route decisions made by Algorithm 4 depend upon the weights $w_{bj}$ given to each efficiency attribute $j$ by each vehicle $b$. In order to find the set of weights which generate the most valuable and reliable routes, a local search of the set of the weights is performed. The choice of step length for each weight $\Delta w_{bj}$ is determined by calculating minimum weight change that changes a single route decision made during an application of Algorithm 4.

Equation (8) [3] 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 20 of Algorithm 4. Here, $\varepsilon$ 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 $i^*$.

$$\Delta w^{i}_{bj} = \frac{\Big(\big(score\,(i^*) - \big(w_{bj}\lambda_{i^*j}\big)\big) - \big(score\,(i) - \big(w_{bj}\lambda_{ij}\big)\big) + \varepsilon\Big)}{\lambda_{ij} - \lambda_{i^*j}} - w_{bj}. \tag{8}$$

Three local search neighbourhoods are considered in the local search algorithm (LS).

- The absolute minimum weight change neighbourhood. This neighbourhood consists of the minimum changes to each weight $w_{bj}$ $\forall b \in B,\ \forall j \in J$, both positive and negative, that change the routes generated by Algorithm 4. This neighbourhood is denoted $\mathfrak{n}_1$.
- The average minimum weight change neighbourhood. This neighbourhood consists of the average of the minimum changes to each weight $w_{bj}$ $\forall b \in B,\ \forall j \in J$, both positive and negative, required to change each route decision to each other possible choice. This neighbourhood is denoted $\mathfrak{n}_3$.
- The allocation order neighbourhood. This neighbourhood is based on applying Algorithm 4 for subsets of vehicles sequentially. The initial implementation of Algorithm 4 is based on determining all vehicle routes simultaneously. A single possible allocation order can be expressed as a permutation of integers which sums to $|B|$. If there are 3 vehicles, then the set of possible allocation orders are: $(1,1,1)$, $(1,2)$, $(2,1)$ and $(3)$. The allocation order $(1,1,1)$ corresponds to applying algorithm 4 to route each vehicle independently in a sequential manner, subtracting the nodes assigned to previous vehicles from the set of non-assigned nodes $U$ each time. The allocation order $(3)$ corresponds to applying algorithm 4 to route all three vehicles simultaneously. In general, the set of possible allocation orders is found by enumerating the set of permutations of combinations integers which sum to $|B|$. This neighbourhood is denoted $\mathfrak{n}_3$.

Finally, let the set of local search neighbourhoods be denoted by $\mathfrak{N}$, i.e.: $\mathfrak{N} = \{\mathfrak{n}_1, \mathfrak{n}_2, \mathfrak{n}_3\}$.

The multi-start local search is the same as that introduced in [3] and is based on the three local search neighbourhoods defined earlier in this section. In each iteration, a local search neighbourhood is selected at random according to a probability distribution $P^{\,local\ search} = \left(\frac{1}{3}, \frac{1}{3}, \frac{1}{3}\right)$. The selected neighbourhood is evaluated by implementing each neighbouring solution in that neighbourhood using Algorithm 4, which uses the machine learning module to predict edge travel times. Such an approach reduces extensive use of the computationally demanding simulation model. The best neighbouring solution is then tested in the simulation model to reveal both a more accurate estimate of the objective value of the neighbouring solution and data regarding the actual road traversal times. The generated data is used to train the machine learning algorithm. Following this the multi-start local search algorithm checks whether a new best overall solution has been identified, if so, it is updated. If not, the current neighbourhood is temporarily removed from the set $\mathfrak{N}$. For example, if the removed neighbourhood is $\mathfrak{n}_1$, then the neighbourhood distribution is updated to $P^{\,local\ search} = \left(0, \frac{1}{2}, \frac{1}{2}\right)$. If all neighbourhoods have been ruled out, the algorithm restarts from a randomly selected set of weights. The restart mechanism considered in this work are described in Section 6.4. See [3] for full details of the multi-start local search algorithm.

## 6.3 Promising initial solution look-up-table learning mechanism

This section describes a mechanism for learning strong partial solutions which can be used as initial solutions when applying Algorithm 4 to generate vehicle routes. A partial solution is an incomplete routing solution corresponding to an intermediate stage of an application of Algorithm 4. Partial solutions are parameterised according to two statistics. Firstly, according to the earliest time ($\phi$) at which a vehicle is expected to reach the last assigned node in their incomplete tour, this information is estimated when testing candidate solutions in the simulation model. Secondly, according to the total number of visited nodes ($\psi$) in the partial solution. Every time a node is added to a solution in a new partial solution is generated. The values of $\phi$ and $\psi$ are used to identify a position in a look-up-table ($L$) of partial solutions. If there are $\Psi$ nodes-visited intervals and $\Phi$ time intervals in the look-up-table, then the relevant element of the look-up-table is $L\left(\lfloor \Phi \frac{\phi}{|N|} \rfloor, \lfloor \Psi \frac{\psi}{T_{max}} \rfloor\right)$. If there is an existing entry for element

$(\chi, \psi)$, the existing entry is replaced only if the new partial solution has a higher final objective value than that of the existing entry. The look-up-table of partial solutions is updated as part of the "Machine learning module" block of Figure 1.

## 6.4 Five variant optimisation modules

This section presents the details of the 5 optimisation modules, that will be the focus of computational experiments in Section 8, including parameter settings.

1. *Local Search* (LS). This optimisation module is based on performing a local search of the efficiency attribute weights $w_{bj}$, as described in Section 6.2. Upon restart, two methods are considered (with equal probability) for generating a new set of random weights. The first method is to generate random values according to a $uniformRandom(-1,1)$ distribution. The second method is to select two sets of weights, from an descending objective value sorted list of the sets of weights tested in previous learnheuristic iterations, according to a geometric distribution, with $\beta$ of the equation of line 25 of Algorithm 4 set to 0.5. In addition, a new random seed ($\sigma$) and a greediness parameter ($\beta \in [0.5, 1]$) is randomly selected upon each restart.

2. *Biased randomisation* (BR). This optimisation module is based on fixing $w_{b5} = 1 \; \forall b \in B$ and $w_{bj} = 0 \; \forall b \in B, \; \forall j \in \{1..\Lambda\} \setminus \{5\}$. This particular set of weights represents the most natural greedy heuristic based on Algorithm 4, i.e., selecting nodes with the highest ratio of reward to travel time. This optimisation module is based on the observation that Algorithm 4 is a biased randomisation algorithm for cases where the geometric distribution parameter $\beta < 1$. In each learnheuristic iteration, Algorithm 4 is applied $\mu$ times. For each application, a new random seed $\sigma$ is sampled, a (greediness) geometric distribution parameter ($\beta \in [0.4, 0.7]$) is selected (values that worked well) and an allocation order is selected at random. After $\mu$ repeated applications, the route with the highest total reward is tested in the simulation model, resulting in additional training data for the machine learning module. $\mu = 10$ was found to provide the best performance.

3. *Biased randomisation with look-up-table learning mechanism* (BRWL). This optimisation module is identical to the BR optimisation module with the exception that it utilises the look-up-table learning mechanism described in Section 6.3. In each application of Algorithm 4 a partial solution is randomly selected as the initial solution with probability $\left(\frac{time}{timeLimit}\right)^2$, that is, the probability of using stored partial solutions as the initial solution gradually increases over the course of the learnheuristic algorithm.

4. *Local search followed by biased randomisation* (LSBR). This optimisation module combines the LS and BR optimisation modules in a sequential manner. The LS optimisation module is used as described above for $\frac{timeLimit}{2}$ of the learnheuristic runtime. Following this, the BR optimisation module is used for the remaining $\frac{timeLimit}{2}$ of the learnheuristic runtime. The implementation of BR differs slightly from that described above. In particular, in this case, BR uses the weight sets and allocation order pairs that were the strongest candidate solutions in each learnheuristic iteration of the LS phase. This approach permits a greater exploration of the weights and allocation order pairs identified as strong candidates in the LS phase.

5. *Local search followed by biased randomisation with look-up-table learning mechanism* (LSBRWL). This optimisation module is identical to the LSBR optimisation module with the exception that the BR phase utilises the look-up-table learning mechanism described in Section 6.3. The look-up-table of partial solutions is updated in both the LS and BRWL phases.

# 7 Learnheuristic Algorithms

An instance of a learnheuristic is defined by its constituent combination of an optimisation module, a machine learning module and simulation model. In Section 8, a comparison is made of the set of learnheuristics composed from all combinations of the machine learning modules described in Section 5 and the optimisation modules described in Section 6. The simulation model in each learnheuristic instance is the traffic simulation described in Section 4. An instance of a learnheuristic algorithm is also defined by a time limit ($timeLimit$), which determines the number of learnheuristic iterations (cycles of the Figure 1 flowchart) that can be completed, and the number of simulation runs, denoted $m$, which are used to test the new candidate solution which is generated in each learnheuristic iteration.

Algorithm 5 outlines the general schema of a learnheuristic algorithm.

**Algorithm 5** Learnheuristic procedure

1: Inputs: Optimisation module, simulation model, machine learning algorithm, $timeLimit$.
2: $bestSol = \emptyset$, $bestObj = -\infty$, $time = 0$
3: **while** $time < timeLimit$ **do**
4: 1. Use the optimisation module (Section 6) to generate a new promising solution using machine learning algorithm predictions (Section 5) of dynamic or solution dependent parameters.
5: 2. Test the new solution in the simulation model (Section 4) to obtain an estimate of its objective value and data for training the machine learning algorithm.
6: 3. Check if the new solution is the best one found so far and update $bestSol$ and $bestObj$ if it is.
7: 4. Train the machine learning algorithm using the data collected so far.
8: **end while**
9: Output: $bestSol$

$m$ is referred to as the size of the *training set* of simulation runs. The random number generator of the $k^{th}$ repeat simulation run out of $m$ is reset using the random seed $k+\kappa$, where the value of $\kappa$ defines a distinct training set. The particular choice of the value of $m$ has a direct influence on the reliability and value of the final solution—low values lead to a risk of overfitting a solution to a small test set. The final solution is then tested in a larger set of repeat simulation runs (of size $\Omega$, see Objective 1), referred to as the *test set* of repeat simulation runs, which reveals the true value of the final solution, as well as a more accurate measure of that solution's reliability level. In Section 8.3, results are provided regarding the effect of the size of the training set on the expected value and the true value of a final solution, as well as the associated reliability level. The reliability level of a solution is calculated as follows:

$$reliability = \frac{\sum_{\omega \in \Omega} \sum_{b \in B} g_{\omega b}}{|\Omega||B|}. \tag{9}$$

Equation 9 defines the reliability level of a solution as the overall rate at which individual routes are successfully completed within the time limit over the course of $\Omega$ repeat simulation runs.

## 7.1 Comparison of optimisation modules on deterministic benchmark TOP instances

Since there are no current benchmark solutions for the problem considered in this work, the optimisation modules are evaluated in isolation, on a standard set of deterministic benchmark problems ([8]) for the TOP. Table 1 provides overall average % gaps (relative to the best known upper bounds), average absolute time gaps and total number of optimal solutions for each optimisation module presented in Section 6.4. These results are provided for each of the 6 subsets of instances considered by [15] as well as overall. 1000 iteration of each method were performed. Table 1 shows that LS and the optimisation modules which make use of LS attain the best results in terms of % gap whilst being only marginally slower than the optimisation modules which do not. The reason

| | LS (25 optimal) | | BR (2 optimal) | | BRWL (4 optimal) | | LSBR (23 optimal) | | LSBRWL (25 optmial) | |
|---|---|---|---|---|---|---|---|---|---|---|
| | % gap | Avg time difference | % gap | Avg time difference | % gap | Avg time difference | % gap | Avg time difference | % gap | Avg time difference |
| 1 | 2.49 | -4.59 | 3.78 | -4.98 | 2.56 | -5.03 | 1.89 | -4.17 | 1.26 | -4.45 |
| 3 | 1.54 | -27.58 | 2.64 | -28.01 | 2.66 | -28.10 | 1.68 | -27.22 | 1.38 | -27.36 |
| 4 | 8.16 | -2478.90 | 11.19 | -2482.49 | 8.60 | -2483.95 | 7.68 | -2476.00 | 6.39 | -2479.08 |
| 5 | 0.73 | -958.34 | 3.98 | -960.20 | 2.17 | -960.57 | 0.79 | -957.43 | 0.70 | -957.88 |
| 6 | 0.52 | -249.69 | 5.88 | -251.03 | 4.60 | -251.38 | 0.35 | -248.67 | 0.44 | -249.08 |
| 7 | 5.56 | -1797.08 | 9.02 | -1799.73 | 6.54 | -1800.37 | 5.86 | -1795.65 | 4.75 | -1796.39 |
| Avg | 3.21 | -1198.45 | 6.27 | -1200.51 | 4.48 | -1201.09 | 3.18 | -1197.14 | 2.64 | -1198.10 |

Table 1: Average % optimality gaps and average solution time differences for subsets of [8] instances

for this is that the LS based methods makes marginally more use of Algorithm 4, which is because, during parameter tuning, it was found to be beneficial to use a moderate number of repeat applications of Algorithm 4 per learnheuristic iterations for BR and BRWL, since this enhanced the diversity of the search of those optimisation modules. The LS based optimisation modules attain between 23 and 25 optimal solutions out of a total of 79 problem instances, and % gaps between 2.64% and 3.21%. The results of Table 1 also show that the look-up-table learning mechanism helps to improve solution quality, this can be seen be comparing the results for BR and BRWL and also the results for LSBR and LSBRWL. In general, Table 1 shows that the optimisation modules upon which the following learnheuristic experiment results are based are fast and competitive algorithms.

# 8 Experiment Results

In this section, all combinations of the optimisation modules and machine learning modules are tested on a number of instances of the TOP set within a traffic simulator. Each combination has a time limit of 30 minutes. The training set size ($m$) is set to 10, Section 8.3 shows this to be sufficient for generating solutions with a reliability level of around 95% when tested on a test set of simulation runs of size 1000.

The experiments were implemented as single threaded Java applications on a core i5 2.8Ghz CPU with 8Gb RAM.

## 8.1 Test Instances

Experiments are carried out based on two different network structures: 1) a randomly generated road network; and 2) a Manhattan road network structure. In both cases, an iterative application of the minimum spanning tree was used to generate a partially connected graph. The minimum spanning tree algorithm proceeds by successively adding arcs connecting the nearest pair of non-connected nodes, until all nodes are connected. The minimum spanning tree algorithm is applied 6 times in a row. In each subsequent application, the arcs added in previous application are removed from consideration, so that only new arcs are added to graph. All of the arcs added to the graph over the 6 applications of the spanning tree algorithm are used as the roads of the network. The resulting road network structures, for the case of the randomly generated and Manhattan road networks, are displayed in Figures 7 and 8 respectively, and are available online [54]. The limited connectivity of the road networks displayed in Figures 7 and 8 give rise of traffic bottlenecks.

The traffic demand patterns of other vehicles are now described. In both examples, the traffic demand intensity of other vehicles follows a triangular distribution with respect to time of day, with a peak demand intensity occurring in the middle of the day. The peak demand rate, $E$, is set to 0.005 per interval, per possible shortest

path route. That is, in each $\Delta t$ time interval of the simulation, there is a probability $p$ that a vehicle initiates a vehicle travelling between each and every pair of nodes. In the first half of the day $p = \frac{2tE}{T_{max}+runIn}$, where $t$ is the current time, $T_{max} + runIn$ is the length of a day, and $E$ is the peak demand. In the second half of the day $p = 2E\left(1 - \frac{t}{T_{max}+runIn}\right)$. In this work $T_{max} = 90$, $runIn = 0$ and $\Delta t = 0.2$. Figures 7 and 8 indicates the relative demand levels on each road using line thickness. It can be seen that for the case of Figure 7 there is a clear traffic bottleneck in the middle of the road network, displayed as a single central node that is connected to a set of roads that each have relatively high traffic demand levels.

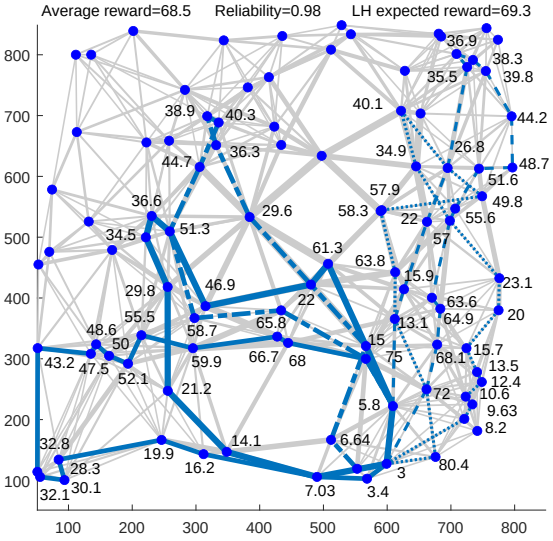


Figure 7: Plot of the best solution for example 1.

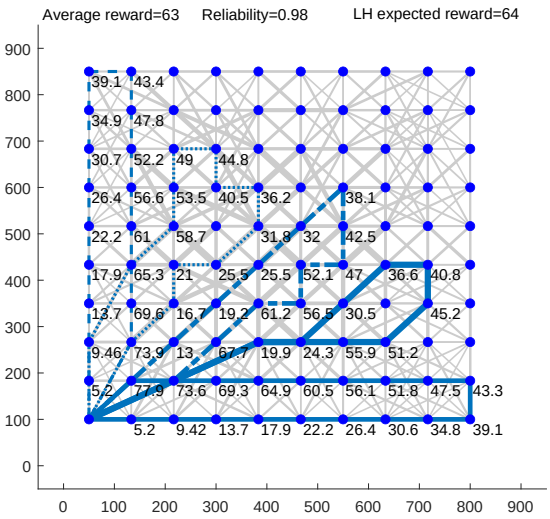


Figure 8: Plot of the best solution for example 2.

This study uses synthetic instances for evaluating the proposed methodology. For the case of real world applications of the proposed methodology, maps of actual road networks can be imported and any traffic simulation can be used. In fact, the role of the simulation module can be played by reality itself, in an, albeit, very slow learnheuristic algorithm. In real problems there may also be multiple demand peaks during a day, driver behaviour and vehicle characteristics may also be more varied. For new problems, it will be advisable to perform a fresh search of the optimisation and learning algorithm hyper-parameters.

## 8.2 Discussion of results

Figure 9 provide: i) test set average reward values; ii) reliability levels; iii) training set average reward values; and iv) root mean squared error (RMSE) values (as used in [55]), for the best solutions found for each instance of a learnheuristic algorithm, tested on the random road network. The RMSE error values are based on comparing machine learning algorithm travel time predictions with the simulation generated travel times. In particular, a squared error is calculated for each route segment in each repeat run of the simulation during a learnheuristic algorithm, where a route segment is defined as the sequence of roads along the shortest path between each pair of consecutive nodes assigned to a vehicle.

Figure 9 (top-left) shows that the best overall solution methods, based on 95% confidence intervals (see error bars), were LSBR-NN, LS-PWL and LSBRWL-PWL. Figure 9 (top-left) shows that the choice of machine learning module had a larger impact on the test set reward values compared to the choice of optimisation module. In particular, OMin and NL lead to very low test set reward values. The choice of machine learning algorithm is so important because, it determines the accuracy of the information used by the optimisation algorithm to generate a solution, and it determines the number of learnheuristic iterations that can be performed within the time limit. Figure 9 (top-right) shows that OMin and NL based learnheuristics also lead to solutions with very low relia-

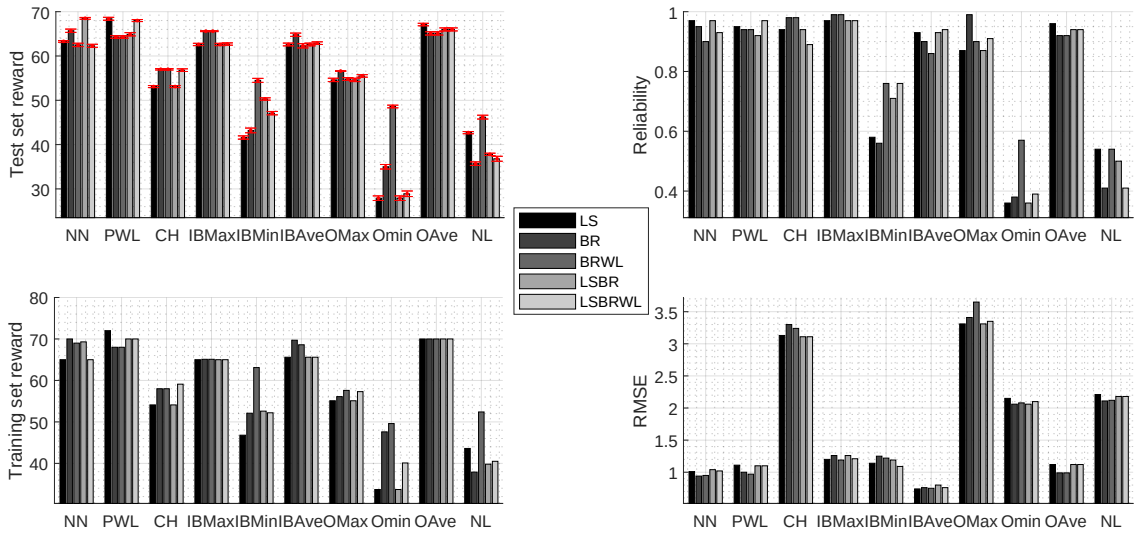


Figure 9: Comparison of all combinations of optimisation and machine learning modules in terms of test set reward, reliability, training set reward, number of nodes visited and RMSE.

bility. These observations can be attributed to the optimistic (underestimated) travel time predictions that these machine learning modules generate. In contrast, IBMax, which generates pessimistic (overestimated) travel time predictions, led to average quality but very reliable solutions. The most effective machine learning modules, in decreasing order of average test set reward, were PWL, OAve, NN and IBMax. OAve is the simplest machine learning module, so it is surprising that this method was so effective. An explanation for the effectiveness of the OAve machine learning module is as follows. In the early stages of applications of Algorithm 4, when congestion levels were very low, OAve typically generated overestimates for travel time predictions, which were later compensated for with underestimates at peak congestion times. So the overall predicted route times were accurate, which is most crucial for ensuring solution reliability. OAve, PWL, NN and IBMax each had good prediction accuracy as shown by the RMSE results by Figure 9 (bottom-right). IBAve has the lowest overall average RMSE, but is also on of the least efficient machine learning modules. This is because the IBAve machine learning module has to search through all of the collected road traversal data, to identify the nearest neighbour, every time a prediction is required. As a result, fewer learnheuristic iterations can be completed within the time limit, which leads to lower quality best found solutions. In contrast, PWL and NN predictions are generated by evaluating relatively efficient models, allowing PWL and NN based learnheuristics to complete more iterations within the time limit. The NN machine learning module performed best in conjunction with the LSBR optimisation module.

The most effective optimisation module, on average, was BRWL. This can be attributed to this optimisation modules efficient search diversification and intensification mechanisms, which enabled it to find reasonable solutions even when it was used in conjunction with the machine learning modules that performed worst on average. The BR component of BRWL, with its randomisation component, provides search diversification. The WL component of BRWL, which records and uses high quality partial solutions as initial solutions, provides search intensification.

Figure 10 displays the results for the case of the Manhattan road network. The same general conclusions can be drawn from those results. Again, learnheuristics based on NN, PWL and OAve have good performance. The best learnheuristic combinations in this case were obtained using the LS-Ave, LSBR-OAve and LSBRWL-OAve learnheuristics. OAve, being the simplest machine learning module, allows for the largest number of learnheuristic iterations within the time limit.

Figure 11 demonstrates convergence of solution quality in terms of collected reward and the solution reliability as more learnheuristic iterations are performed for the LS-NN learnheuristic. In the early phases, the learnheuristic

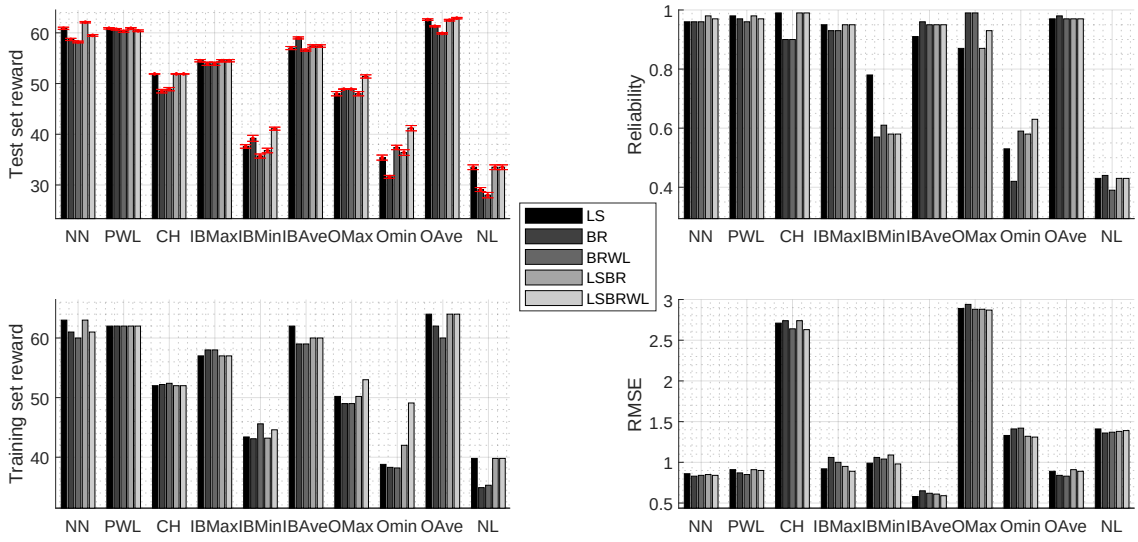


Figure 10: Comparison of all combinations of optimisation and machine learning modules in terms of test set reward, reliability, training set reward, number of nodes visited and RMSE, tested on the random road network.

generates solutions with low reward and reliability, which is because the learnheuristic is only just beginning to learn about road traversal times. Figure 11 shows that reward and reliability are closely related, which is because the rewards collected by vehicles which do not complete their routes with the time limit are lost.

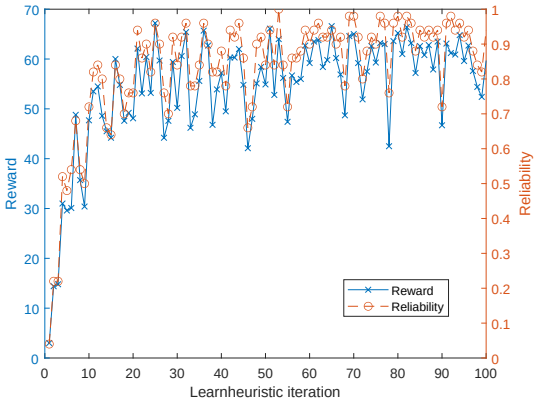


Figure 11: Convergence of reward and reliability as more learnheuristic iterations are performed.

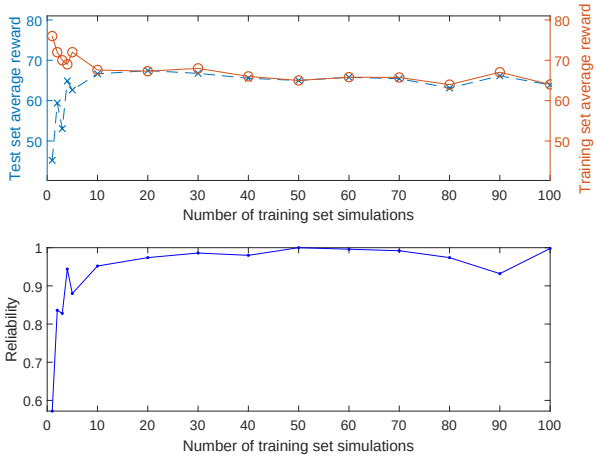


Figure 12: The effect of the size of the training set size on average reward and reliability.

## 8.3 The Effect of Training Set Size

This section examines the effect of the size of the training set of simulation runs, which is used to evaluate the strongest candidate solution from each learnheuristic iteration. Given a time limit, increasing (decreasing) the the training set size reduces (increases) the total number of learnheuristic iterations. Figure 12 (top) shows the effect of the training set size on the average reward, evaluated against both the training (right y-axis) and test (left y-axis) sets, based on the effective LS-PWL learnheuristic. Figure 12 (top) shows that a training set size of at least 10 is required for the average rewards for the training and test sets to converge. For smaller training set sizes overfitting occurs, which is characterised by very high expected rewards when solutions are evaluated against the training set and very low average rewards when solutions are evaluated against the test set. Additionally, Figure 12

(bottom) shows that very small training set sizes lead to solutions with a low reliability level, which is the reason why solutions achieve a low average reward. As the size of the training set is increased, the reliability level of solutions, evaluated against the test set, tends to 100%, with the exceptions of training set sizes of 80 and 90. At the same time the average rewards, evaluated against the training and test sets, decreases slightly. The explanation for this is that larger training set sizes reduce the total number of learnheuristic iterations that are performed. For a training set size of 90 a solution with a reduced reliability level is generated, however, this solution still has a slightly higher average reward. This suggests that the solution for a training set size of 90 is ambitious, in the sense that it aims to collect a large total reward, but does so at the cost of a reduced reliability level.

# 9 Conclusions

This work has considered solutions for the TOP set within a traffic simulation, a problem which gives rise to a combined learning and optimisation problem. A learnheuristic solution approach has been proposed, which integrates simulation, machine learning and optimisation within an iterative framework.

A traffic simulation model was proposed and validated, a model that is based on modelling anticipatory driving behaviour. It was shown that road traversal times are generally strongly dependent upon the time of day and the particular road. As a result, the road traversal time prediction problem, the problem to be addressed by the machine learning module, was decomposed by road and assumed to be a function of the time of day of travel.

Five optimisation module variants were proposed, each based on an efficient event based routing algorithm. Before investigating their effectiveness as components of learnheuristic algorithms, they were compared and validated using a set of standard benchmark deterministic problem instances. One of the most effective was a sensitivity based local search, which optimises the weights given to a number of different next-node selection criteria (efficiency attributes).

This work has investigated the effectiveness of a variety of different machine learning modules. It was found that some machine learning modules have beneficial characteristics with regard to the average reward and reliability level of the final solutions generated. In particular, it was found that piecewise linear regression and neural networks lead to solutions with high average reward values. A machine learning module based on simple averaging was also found to provide consistently good results, when used in conjunction with any of the proposed optimisation modules.

A critical issue that this work has addressed was that of how best to utilise the run time of a learnheuristic algorithm. It was found that, in order to achieve solutions with good performance, as measured against a large test set of simulation runs, a moderate training set size of 10 or greater was required. Increasing the size of the training set beyond 10 improved solution reliability at the expense of reduced average reward. This effect was attributed to the corresponding reduction in the total number of learnheuristic iterations that can be performed.

By considering test instances with features common to many congested road networks, it was demonstrated that the proposed learnheuristic solution approach was capable of generating solutions which avoid highly congested roads at the times when they are most congested. In particular, it was common to try to traverse the most congested routes as early as possible, as to avoid the busiest roads at peak demand times.

Based on this work, future lines of research include the application of learnheuristics to other combinatorial optimisation problems with parameter learning problems. Learnheuristics themselves can be extended to make use of a hierarchy of simulation models, with increasing levels of detail and computational cost. The question of interest in such cases is how best to distribute computation effort between the set of simulation models, within the run time of a learnheuristic. Similarly, learnheuristics can be extended to make use of a combination of machine learning modules, perhaps relying on simple approximation methods when little information is known about the problem under consideration, and switching to more detailed models once they have been trained using simulation data. The consideration of other optimisation modules is also a promising area for future work, for instance, the

best way in which to integrate an evolutionary algorithm within a learnheuristic is an open question. Another direction for future research consists of comparing learnheuristics with other approaches for tackling optimisation problems with inherent learning problems. Another direction is to apply learnheuristics to routing problems with more constraints, such as time windows, electric vehicle charging requirements, pollution minimisation and pick-up and delivery requirements.

## Acknowledgements

This work has been partially supported by the Erasmus+ programme (2017-1-ES01-KA103-036672) and by the Spanish Ministry of Science, Innovation, and Universities (RED2018-102642-T).

## REFERENCES

[1] B.L. Golden, L. Levy, and R. Vohra. "The orienteering problem". In: *Naval Research Logistics* 34 (1987), pp. 307–318.

[2] D. Gavalas et al. "A survey on algorithmic approaches for solving tourist trip design problems". In: *Journal of Heuristics* 20.3 (2014), pp. 291–328.

[3] Christopher Bayliss et al. "A Learnheuristic Approach for the Team Orienteering Problem with Aerial Drone Motion Constraints". In: *Applied Soft Computing* (2020). Accepted for publication.

[4] G. Laporte and S. Martello. "The selective traveling salesman problem". In: *Discrete Applied Mathematics* 26 (1990), pp. 193–207.

[5] A. Gunawan, H.C. Lau, and P. Vansteenwegen. "Orienteering Problem: A survey of recent variants, solution approaches and applications". In: *European Journal of Operational Research* 255 (2016), pp. 315–332.

[6] W. Souffriau et al. "A personalised tourist trip design algorithm for mobile tourist guides". In: *Applied Artificial Intelligence* 22.10 (2008), pp. 964–985.

[7] C. Verbeeck et al. "A fast solution method for the time-dependent orienteering problem". In: *European Journal of Operational Research* 236 (2014), pp. 419–432.

[8] I.-M. Chao, B.L. Golden, and E.A. Wasil. "The team orienteering problem". In: *European Journal of Operational Research* 88 (1996), pp. 464–474.

[9] C. Archetti et al. "The Capacitated Team Orienteering and Profitable Tour Problems". In: *The Journal of the Operational Research Society* 60.6 (2009), pp. 831–842.

[10] Shih-Wei Lin and Vincent F. Yu. "A simulated annealing heuristic for the team orienteering problem with time windows". In: *European Journal of Operational Research* 217 (2012), pp. 94–107.

[11] Claudia Archetti et al. "The Team Orienteering Arc Routing Problem". In: *Transportation Science* 48.3 (2014), pp. 442–457.

[12] Vincent F. Yu et al. "Two-level particle swarm optimization for the multi-modal team orienteering problem with time windows". In: *Applied Soft Computing* 61 (2017), pp. 1022–1040.

[13] S. Butt and D. Ryan. "An optimal solution procedure for the multiple tour maximum collection problem using column generation". In: *Computers and Operations Research* 26 (1999), pp. 427–441.

[14] Morteza Keshtkaran et al. "Enhanced exact solution methods for the Team Orienteering Problem". In: *International Journal of Production Research* 54.2 (2016), pp. 591–601.

[15] Racha El-Hajj, Duc-Cuong Dang, and Aziz Moukrim. "Solving the Team Orienteering Problem with Cutting Planes". In: *Computers and Operations Research* 74 (2016), pp. 21–30.

[16] Duc-Cuong Dang, Rym Nesrine Guibadj, and Aziz Moukrim. “An effective PSO-inspired algorithm for the team orienteering problem”. In: *European Journal of Operational Research* 229.2 (2013), pp. 332–344.

[17] Wouter Souffriau et al. “The Multiconstraint Team Orienteering Problem with Multiple Time Windows”. In: *Transportation Science* 47.1 (2013), pp. 53–63.

[18] Vicente Campos et al. “GRASP with path relinking for the orienteering problem”. In: *The Journal of the Operational Research Society* 65.12 (Dec. 2014), pp. 1800–1813.

[19] S.W. Lin. “Solving the team orienteering problem using effective multi-start simulated annealing”. In: *Applied Soft Computing* 13 (2013), pp. 1064–1073.

[20] Liangjun Ke et al. “Pareto mimic algorithm: An approach to the team orienteering problem”. In: *Omega* 61 (2016), pp. 155–166.

[21] Eleftherios Tsakirakis et al. “A similarity hybrid harmony search algorithm for the Team Orienteering Problem”. In: *Applied Soft Computing* 75 (2019), pp. 130–147.

[22] J. Ferreira, A. Quintas, and J.A. 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.

[23] Gorka Kobeaga, María Merino, and Jose A. Lozano. “An efficient evolutionary algorithm for the orienteering problem”. In: *Computers and Operations Research* 90 (2018), pp. 42–59.

[24] Wanzhe Hu, Mahdi Fathi, and Panos M. Pardalos. “A multi-objective evolutionary algorithm based on decomposition and constraint programming for the multi-objective team orienteering problem with time windows”. In: *Applied Soft Computing* 73 (2018), pp. 383–393.

[25] T. Ilhan, S.M.R. Iravani, and M.S. Daskin. “The orienteering problem with stochastic profits”. In: *IIE Transactions* 40 (2008), pp. 406–421.

[26] J. Panadero et al. “Maximizing reward from a team of surveillance drones under uncertainty conditions: a simheuristic approach”. In: *European Journal of Industrial Engineering* (2020).

[27] Ian H. Witten, Eibe Frank, and Mark A.Hall. *Data Mining: Practical Machine Learning Tools and Techniques*. Third. Burlington, USA: Elsevier, 2011.

[28] A A Juan et al. “A review of simheuristics: Extending metaheuristics to deal with stochastic combinatorial optimization problems”. In: *Operations Research Perspectives* 2.1 (2015), pp. 62–72.

[29] Claudia Archetti and M. Grazia Speranza. “A survey on matheuristics for routing problems”. In: *EURO Journal on Computational Optimization* 2 (2014), pp. 223–246.

[30] Jun Zhang et al. “Evolutionary Computation Meets Machine Learning: A Survey”. In: *IEEE Computational Intelligence Magazine* (Nov. 2011), pp. 68–75.

[31] El-Ghazali Talbi. “Combining metaheuristics with mathematical programming, constraint programming and machine learning”. In: *Annals of Operations Research* 240 (2016), pp. 171–215.

[32] Laetitia Jourdan, Clarisse Dhaenens, and El-Ghazali Talbi. “Using Datamining Techniques to Help Metaheuristics: A Short Survey”. In: *LNCS. Hybrid Metaheuristics*. `https://hal.inria.fr/inria-00269969`. Berlin, Heidelberg: Springer, 2006, pp. 57–69.

[33] Laura Calvet et al. “Learnheuristics: hybridizing metaheuristics with machine learning for optimization with dynamic inputs”. In: *Open Mathematics* 15 (2017), pp. 261–280.

[34] Hasan Dündar, Mine Ömürgönülşen, and Mehmet Soysal. “A review on sustainable urban vehicle routing”. In: *Journal of Cleaner Production* 285 (2021), pp. 654–676.

[35] Panagiotis N. Kechagiopoulos and Grigorios N. Beligiannis. “Solving the Urban Transit Routing Problem using a particle swarm optimization based algorithm”. In: *Applied Soft Computing* 21 (2014), pp. 654–676.

[36] Jalel Euchi and Rafaa Mraihi. “The urban bus routing problem in the Tunisian case by the hybrid artificial ant colony algorithm”. In: *Swarm and Evolutionary Computation* 2 (2012), pp. 15–24.

[37] Erfan Babaee Tirkolaee, Iraj Mahdavi, and Mir Mehdi Seyyed Esfahani. “A robust periodic capacitated arc routing problem for urban waste collection considering drivers and crew’s working time”. In: *Waste Management* 76 (2018), pp. 138–146.

[38] Goran Ćirović, Dragan Pamučar, and Darko Bžanić. “Green logistic vehicle routing problem: Routing light delivery vehicles in urban areas using a neuro-fuzzy model”. In: *Expert Systems with Applications* 41 (2014), pp. 4245–4258.

[39] G. Clarke and J. W. Wright. “Scheduling vehicles from a depot to a number of delivery points”. In: *Operations Research* 12 (1964), pp. 568–581.

[40] Ramin Raeesi and Konstantinos G. Zografos. “The multi-objective Steiner pollution-routing problem on congested urban road networks”. In: *Transportation Research Part B* 122 (2019), pp. 457–485.

[41] D. C. Gazis et al. “Nonlinear follow-the-leader models of traffic flow”. In: *Operational Research* 9 (1961), pp. 545–560.

[42] M.E. Lárraga, J.A. del Río, and L. Alvarez-lcaza. “Cellular automata for one-lane traffic flow modeling”. In: *Transportation Research Part C* 13 (2005), pp. 63–74.

[43] S. KrauB. “Microscopic modelling of traffic flow: Investigation of collision free vehicle dynamics”. PhD thesis. Cologne, Germany: University of Cologne, 1998.

[44] Florian Siebel and Wolfram Mauser. “On the Fundamental Diagram of Traffic Flow”. In: *SIAM Journal on Applied Mathematics* 66.4 (2006), pp. 1150–1162.

[45] Kai Nagel, Peter Wagner, and Richard Woesler. “Still Flowing: Approaches to Traffic Flow and Traffic Jam Modeling”. In: *Operations Research* 51.5 (2003), pp. 681–710.

[46] Javier Ferrer, Manuel López-Ibáñez, and Enrique Alba. “Reliable simulation-optimization of traffic lights in a real-world city”. In: *Applied Soft Computing* 78 (2019), pp. 697–711.

[47] Terry L. Friesz Byung-Wook Wie Roger L. Tobin and David Bernstein. “A Discrete Time, Nested Cost Operator Approach to the Dynamic Network User Equilibrium Problem”. In: *Transportation Science* 29.1 (1995), pp. 79–92.

[48] Marco Günther et al. “Multivalued Fundamental Diagrams and Stop and Go Waves for Continuum Traffic Flow Equations”. In: *SIAM Journal on Applied Mathematics* 64.2 (2003), pp. 468–483.

[49] Richard J. Gibbens and Yunus Saatci. “Data, Modelling and Inference in Road Traffic Networks”. In: *Philosophical Transactions: Mathematical, Physical and Engineering Sciences* 366.1872 (2008), pp. 1907–1919.

[50] Willi-Hans Steeb. *The non-linear workbook*. Singapore, New Jersey, London, Hong Kong: World Scientific, 1999.

[51] D.E. Rumelhart and J.L. McClelland. *Parallel distributed processing: explorations in the microstructure of cognition,* Cambridge, MA: MIT Press, 1986.

[52] Christian Fikar et al. “A discrete-event driven metaheuristic for dynamic home service routing with synchronised trip sharing”. In: *European Journal of Industrial Engineering* 10.3 (2016), pp. 323–340.

[53] Alex Grasas et al. “Biased Randomization of Heuristics using Skewed Probability Distributions: a survey and some applications”. In: *Computers & Industrial Engineering* 110 (2017), pp. 216–228.

[54] Christopher Bayliss. *Test instance data*. https://www.researchgate.net/publication/340732103_LH_Traffic_TOP_Road_Networks. last accessed April 2020. 2020.

[55] P. Lakshmi and M. Abdullah Khan. “Stability enhancement of a multimachine power system using fuzzy logic based power system stabilizer tuned through genetic algorithm”. In: *Electrical Power and Energy Systems* 22 (2000), pp. 137–145.