A Machine Learning-driven Two-phase Metaheuristic for Autonomous Ridesharing Operations

Claudia Bongiovanni[a,d,*], Mor Kaspi[b], Jean-François Cordeau[c], Nikolas Geroliminis[d]

[a] *Transport and Mobility Laboratory*
School of Architecture, Civil and Environmental Engineering
École Polytechnique Fédérale de Lausanne (EPFL)
CH-1015 Lausanne, Switzerland
[b] *Analytics for Urban Transportation & Operations Laboratory*
Department of Industrial Engineering
Tel-Aviv University
Tel-Aviv, 69978, Israel
[c] *HEC Montréal*
3000 chemin de la Côte-Sainte-Catherine,
Montréal, H3T2A7, Canada
[d] *Urban Transport Systems Laboratory*
School of Architecture, Civil and Environmental Engineering
École Polytechnique Fédérale de Lausanne (EPFL)
CH-1015 Lausanne, Switzerland

Abstract

This paper contributes to the intersection of operations research and machine learning in the context of autonomous ridesharing. In this work, autonomous ridesharing operations are reproduced through an event-based simulation approach and are modeled as a sequence of static subproblems to be optimized. The optimization framework consists of a novel data-driven metaheuristic approach composed of two phases. The first phase consists of a greedy insertion heuristic that assigns new online requests to vehicles. The second phase consists of a local-search based metaheuristic that iteratively revisits previously-made vehicle-trip assignments through intra- and inter-vehicle route exchanges. These exchanges are performed by selecting from a pool of destroy-repair operators using a machine learning approach that is trained offline on a large dataset composed of more than one and a half million examples of previously-solved autonomous ridesharing subproblems.

Computational results are performed on multiple dynamic instances extracted from real ridesharing data published by Uber Technologies Inc. Results show that the proposed data-driven optimization approach outperforms benchmark state-of-the-art metaheuristics by up to about 28 percent, on average. Managerial

*Corresponding author
Email addresses: `claudia.bongiovanni@epfl.ch` (Claudia Bongiovanni), `morkaspi@tauex.tau.ac.il` (Mor Kaspi), `jean-francois.cordeau@hec.ca` (Jean-François Cordeau), `nikolas.geroliminis@epfl.ch` (Nikolas Geroliminis)

insights highlight the correlation between selected vehicle routing features and the performance of the metaheuristics in the context of autonomous ridesharing operations.

Keywords: dial-a-ride problem, electric autonomous vehicles, online optimization, large neighborhood search, metaheuristics, machine learning

1. Introduction

Urban mobility currently faces critical performance challenges in view of continued urban population growth. Technological advances, such as autonomous mobility and internet communication technologies, enable the development of innovative mobility services, such as autonomous ridesharing. The latter can flexibly adapt to changing transportation conditions and needs, such as dynamic demand and congestion patterns. However, these services come with additional operational challenges that must be carefully considered by transportation agencies and operators (Hyland & Mahmassani, 2020).

The use of electric autonomous vehicles (e-AVs) is expected to enhance ridesharing operations by offering several new opportunities. First, e-AVs provide greater flexibility than human-guided vehicles to efficiently divert their routes as frequently as desired during operations (Ferrucci & Bock, 2015). Second, e-AVs can operate non-stop, being unbound from driver shifts. This feature may help save vehicle deadhead miles to decentralized depots and provide higher service levels. Third, the type of service provided by e-AVs does not need to be pre-defined (e.g., taxi service, ridesharing service, public transport) and can instead automatically adapt in response to demand needs.

Despite its many advantages, the use of e-AVs for ridesharing operations poses numerous challenges. From an operational perspective and in order to take full advantage of the capabilities of e-AVs, the planning process must be efficiently re-optimized as new information arrives (e.g., new demand, congestion level, vehicle breakdowns). This includes vehicle routing decisions, scheduling decisions, decisions regarding detour to charging stations, charging times, and depot relocation decisions. The combination of all these features undoubtedly composes a complex decision-making problem to be optimized, especially for dynamic autonomous ridesharing operations, since decisions must be made in a very limited computational time.

Ridesharing demand shows repeated patterns over time, although stochasticity occurs in daily and seasonal trends (Liu et al., 2019). As such, ridesharing providers face similar vehicle routing problems over time. These problems share multiple similarities in terms of demand distribution, vehicle states, and route planning. Assuming that problems with similar characteristics can be optimized using similar methods, information obtained from problems solved in the past can become useful for solving new problems in the future. As shown in Larsen et al. (2021), historical information may support the solution process of complex decision-making problems. As such, there is a need to develop new data-driven optimization approaches that leverage historical information to improve real-time autonomous ridesharing operations.

This paper introduces a data-driven optimization approach to solve the dynamic electric autonomous dial-a-ride problem (Bongiovanni et al., 2019, e-ADARP,), in which a fleet of e-AVs provides service to unknown online requests. Dynamic e-ADARP operations are modeled as a sequence of static e-ADARP subproblems to be optimized with the arrival of online requests. Given the hard operational constraints imposed by the e-ADARP, requests may be rejected. As such, the goal of the subproblems consists of a weighted objective function maximizing the number of accepted requests, while minimizing total operational cost and user service dissatisfaction.

Realistic operations of the dynamic e-ADARP are represented through an event-based simulation framework. Each static e-ADARP subproblem is solved through a two-phase metaheuristic approach. The first phase is applied to myopically insert new requests into the vehicle plans and employs an efficient scheduling algorithm for the e-ADARP (see Chapter 3 in Bongiovanni, 2020). The second phase phase is applied to iteratively revisit decisions taken in the first phase through intra- and inter- vehicle route exchanges. Such exchanges are performed by employing common destroy-repair operators from the vehicle routing literature in the context of a large neighborhood search (LNS, Pisinger & Ropke, 2010). Differently from adaptive large neighborhood search (ALNS, Ropke & Pisinger, 2006), destroy-repair operators are selected through a machine learning-driven approach exploiting historical information on previously-solved e-ADARP problems. Historical e-ADARP information is constructed offline through simulation and consists of a large dataset composed of more than one and a half million examples on the performance of the competing destroy-repair operators.

Numerical experiments are performed on real ridesharing data from Uber Technologies Inc. The proposed machine learning-based large neighborhood search, denoted by MLNS, is compared to selected state-of-the-art large neighborhood search approaches. Results on multiple static e-ADARP subproblems show that the machine learning component can increase the expected quality of solutions by up to about 28 percent with respect to a simple large neighborhood search and by up to about nine percent with respect to adaptive large neighborhood search (ALNS, Ropke & Pisinger, 2006). Managerial insights highlight the importance of selected vehicle routing features when deciding on the destruction-repair operators to employ.

The rest of the paper is organized as follows: Section 2 provides a review of the relevant literature and provides a summary of the main methodological contributions, Section 3 describes the dynamic e-ADARP, Section 4 defines the solution approach, Section 5 focuses on the ML approximation, Section 6 describes the data generation approach, Section 7 presents the extracted problem features for the dynamic e-ADARP, and Section 8 contains the results of the numerical experiments. A summary and future directions are provided in Section 9.

2. Literature review

This section introduces the dial-a-ride problem, its dynamic variants, and its solution methodologies. Specifically, it focuses on two-phase heuristic solution methodologies and large neighborhood search, as this is the framework employed in this study. The section concludes by presenting recent literature combining the field of machine learning and operations research and by providing the literature gaps covered by this study.

2.1. The dial-a-ride problem

In the standard dial-a-ride problem (DARP), minimum cost routes and schedules are defined for a fleet of vehicles exiting known depots and serving a set of pre-booked customers with given pickup and dropoff locations (Cordeau & Laporte, 2007). The optimization can take into consideration multiple criteria and can include multiple types of users and destination depots (e.g., Braekers et al., 2014; Parragh, 2011). Typical operational constraints integrate vehicle capacity, maximum route duration, and maximum user ride time constraints. In addition, service start times at pickup and dropoff locations are usually limited by time window constraints. In a recent work, Bongiovanni et al. (2019) have generalized the standard DARP by considering the use of electric autonomous vehicles (e-ADARP). Given that autonomous vehicles can operate non-stop, the e-ADARP relaxes route duration constraints but integrates new operational constraints including decisions regarding detours to charging stations, recharging times, and selection of destination depots. The goal of the e-ADARP is to minimize a weighted-sum objective function consisting of the total vehicle travel time and extra user ride time. For a review on DARP studies, the reader is referred to Ho et al. (2018).

The DARP literature can be divided into two main streams, namely static and dynamic DARP. In the first case, demand is fully known in advance and needs to be served, whereas, in the second case, demand is revealed online. No information about future requests is typically assumed, although some stochastic information (e.g., Albareda-Sambola et al., 2014; Ichoua et al., 2006) and forecasts (e.g., Zalesak & Samaranayake, 2021; Peled et al., 2019; Alonso-Mora et al., 2017; Ferrucci & Bock, 2016; Ferrucci et al., 2013) may be used. Given the inherent uncertainty about demand, requests are allowed to be rejected in the dynamic DARP. One of the main challenges affecting dynamic DARP algorithms concerns finding the right trade-off between solution quality and computing time, which is typically very limited given that decisions should be taken in real-time. In this work we model dynamic e-ADARP opertions as a sequence of static e-ADARP subproblems to be optimized. Such subproblems need to be efficiently solved while respecting constraints imposed by the electric and autonomous nature of the fleet, other than classic dial-a-ride features. For a review on dynamic vehicle routing problems, the reader can refer to Berbeglia et al. (2010).

2.2. Two-phase heuristic approaches

Two-phase heuristic approaches have been widely applied to solve a variety of tightly-constrained vehicle routing problems (e.g., Archetti et al., 2018; Schilde et al., 2011b; Parragh et al., 2009). The first phase typically attempts to assign requests to vehicles through a polynomial-time insertion heuristic (Jaw et al., 1986). The assignment needs to be feasible and thus respect all of the constraints imposed by the treated problem. For each request, multiple request-vehicle assignments may be feasible. Furthermore, multiple routing options may exist for the same request-vehicle assignment. Hence, there is a computationally prohibitive number of pickup-dropoff insertion tentatives to be performed. This number can be reduced by considering time window and ride-time considerations (e.g., Coslovich et al., 2006; Diana & Dessouky, 2004) or by inserting the pickup and dropoff of the respective request one by one (e.g., Parragh et al., 2009). The computational burden can be further reduced by considering parallel insertion heuristics (e.g., Potvin & Rousseau, 1993). The lowest-cost solution (e.g., Marković et al., 2015) or the k-lowest cost solution (e.g., Diana & Dessouky, 2004) is finally selected out of all feasible insertions. For the e-ADARP, the cost associated with the insertion of pickup and dropoff requests can be evaluated by the scheduling algorithm proposed in Chapter 3 in Bongiovanni (2020). The goal of the scheduling algorithm is to be able to find minimal excess-time and battery-feasible schedules for fixed routes.

The second phase is designed to re-optimize previous insertions through sequential inter- and intra-vehicle request exchanges and charging plan modifications. The route adjustments are performed by sequentially destroying and repairing part of the vehicle plans, thus following a local search-based approach.

2.3. Local-search based metaheuristics

Local Search (LS) or local search-based metaheuristics are popular optimization approaches for vehicle routing problems (Funke et al., 2005). These methods iteratively transform a given solution through elementary neighborhood moves to improve the objective function (e.g., Beojone & Geroliminis, 2021; Bertsimas et al., 2019; Simonetto et al., 2019). Each elementary move is composed of a simple heuristic shuffling a limited number of routing decisions, e.g., 2-opt. Large Neigborhood Search (LNS) extends the local search paradigm by considering moves shuffling a larger share of routing decisions. Originally introduced by Shaw (1997), LNS has been widely applied to solve large-scale static (e.g., Molenbruch et al., 2017; Masmoudi et al., 2016; Parragh et al., 2010) and dynamic (e.g., Schilde et al., 2014, 2011a; Attanasio et al., 2004) transportation problems. For a review of LNS, the reader is referred to Section 13.2 in Pisinger & Ropke (2010).

In LNS, neighborhood moves may be performed through a complex neighborhood structure, which sequentially destroys and repairs a given solution. The destruction degree is typically drawn from a uniform distribution supported on a bounded interval. If the bound is too restrictive, the incumbent solution may be modified only marginally. As a result, the search may have difficulties in moving towards promising neighborhoods and get trapped in a local minimum. One way of dealing with this drawback is to loosen the interval bound, allowing the algorithm to re-arrange a higher percentage of the vehicle requests. Nevertheless, higher destruction degrees typically have a negative effect on computational time (Pisinger & Ropke, 2010). Another methodology which is frequently employed to try and escape local minima explores non-improving and infeasible solutions (e.g., Cordeau & Laporte, 2003; Cordeau et al., 2001). Worsening solutions may be generally explored by employing an acceptance criterion based on simulated annealing (SA, Nikolaev & Jacobson, 2010). Simulated annealing employs an analogy with metallurgy by using a probability function that depends on an initial temperature and a cooling rate controlling the likelihood of accepting deteriorating solutions over successive iterations. If worsening solutions are accepted, the solution is expected to iteratively recover towards promising areas of the search space. Such recovery cannot be guaranteed as it highly depends on the type of operator and destruction degree being employed. The recovery is finally bounded by a maximal number of successive non-improving steps, which may determine an early exit from the search.

Recent literature has demonstrated added value in switching between multiple destroy-repair mechanisms during the search by adopting a metaheuristic approach (Elshaer & Awad, 2020). Metaheuristics employ a selection mechanism to automate the choice of the destroy-repair methods to be employed at any iteration during the search. Metaheuristics are frequently also noted as hyper-heuristics (e.g., Burke et al., 2013), although the second typically refers to a combination of metaheuristics rather than destroy-repair methods.

Multiple recent studies from the vehicle routing literature have employed a selection mechanism composed of a score function providing a measure for the success of selected destroy-repair operators on previous iterations (e.g., Sacramento et al., 2019; Gschwind & Drexl, 2019; Li et al., 2016; Goeke & Schneider, 2015). In particular, the score of each destroy-repair couple is initialized according to a "roulette wheel" mechanism and it is updated through a linear function which employs statistics on the solution quality of the produced moves during previous iterations. The update magnitude is scaled by a reaction factor r, controlling how quickly the linear function reacts to changes in the statistics. This selection mechanism, introduced in Ropke & Pisinger (2006), is denoted by adaptive large neighborhood search (ALNS). For a review of ALNS, the reader is referred to Section 13.2.1 in Pisinger & Ropke (2010).

In ALNS, a warm-up time of hundreds of iterations is typically needed to tune the several destroy-repair scores before being able to make statistically-informed choices between competing operators. Indeed, ALNS is typically employed for large static problems, which are not tightly bound by computational time. In this work, we propose an alternative approach, i.e., MLNS, that allows to eliminate the warm-up time required to tune the adaptive mechanism and that is more suitable for real-time settings.

2.4. Machine learning for operations research problems

Recent advances in artificial intelligence have fostered new research directions at the intersection between machine learning (ML) and optimization. While optimization methods have been often employed in ML algorithms (e.g., Bertsimas & Dunn, 2017; Gunluk et al., 2016; Bertsimas & Shioda, 2007), the use of ML in optimization algorithms is relatively new and has been focusing on the following tasks: (1) ML as an approximation tool to tackle time-consuming tasks in discrete optimization algorithms (e.g., Zhang et al., 2021; Bonami et al., 2018; Kruber et al., 2017; Lodi & Zarpellon, 2017), (2) ML to heuristically solve discrete optimization problems (e.g., Larsen et al., 2021; Bengio et al., 2021; Vinyals et al., 2015), and (3) ML to choose among a number of competing algorithms to solve hard optimization problems (e.g., Chen & Tian, 2019; Kerschke et al., 2019; Malitsky et al., 2013; Gomes & Selman, 2001). Specifically, the last task is also known as meta-learning for algorithm porfolios and has focused on automating the selection of solution frameworks by extracting significant problem features for a number of discrete combinatorial problems. Finally, increasing interest has been shown in the use of machine learning to learn LNS-based heuristics, mostly focusing on repair operators (e.g. Wu et al., 2021; Gao et al., 2020; Hottung & Tierney, 2019; Lu et al., 2019). However, to the best of our knowledge, there is still a lack of knowledge about the relationship between problem characteristics and the selection process for popular handcrafted LNS-based heuristics.

2.5. Literature gap

This work proposes a machine learning-driven metaheuristic, i.e. MLNS, to efficiently select destroy-repair heuristics during the search. The selection process is based on iteration-specific vehicle routing characteristics of the problem being optimized. The idea is to exploit past information, i.e., from previously solved similar problems, to guide the search and quickly generate good solutions, i.e., avoiding a warm-up period. Other than driving the search, the combined use of machine learning within metaheuristics allows to identify features of the dial-a-ride problem (and more generally vehicle routing problems) that impact the solution process and to classify new problem instances accordingly.

In MLNS, the prediction task replaces the roulette wheel mechanism proposed in ALNS (Ropke & Pisinger, 2006) and is based on a large dataset containing examples of moves from all of the competing LNS algorithms on several instances and routing solutions. The proposed optimization framework fundamentally differs from ALNS. Specifically, ALNS assumes that the future of the search should be driven by the success of the competing operators in previous iterations, while MLNS assumes that the future of the search should be driven by estimated algorithm performances at the current iteration (which are extracted from previously-solved e-ADARP subproblems).

In this work, the prediction task is tackled through ensemble learning, namely random forest (RF) classification (Breiman, 2001). Random forests have been applied to improve the solution process of several operations research problems in transportation science and supply chain (e.g. Nabian et al., 2019; Jun et al., 2019; Wu et al., 2017; Bonfietti et al., 2015). They have a similar performance compared to other popular machine learning methodologies, such as support vector machines, while having numerous practical advantages. What makes RF suitable for this work is its ability to automatically select features, provide a variable importance measure, handle outliers and very big datasets. Note that several methodologies may be employed and compared for the prediction task, even within the machine learning field. However, a thorough comparison between prediction methodologies does not lie within the scope of this study.

3. Problem description

The dynamic e-ADARP can be seen as a sequence of static e-ADARP subproblems to be optimized. This Section provides a summary of the static e-ADARP, proposed in Bongiovanni et al. (2019), and highlights its main adaptations for use in dynamic e-ADARP settings.

3.1. Static e-ADARP

The static e-ADARP considers a fleet of e-AVs $\mathcal{K} = \{1, \ldots, k\}$, leaving from origin depots $o^k \in \mathcal{O}$ to provide service to a set of requests $i \in \{1, \ldots, n\}$. The requests are characterized by given pickup locations $\mathcal{P} = \{1, \ldots, n\}$, dropoff locations $\mathcal{D} = \{n+1, \ldots, 2n\}$, and time windows $[arr_i, dep_i]$ around the desired pickup times or dropoff times. Each request $i \in \{1, \ldots, n\}$ is represented by a load l_i (i.e., $l_i \geq 0$ for $i \in \mathcal{P}$ and $l_i = -l_{i-n}$ for $i \in \mathcal{D}$), a service time d_i, and a maximum ride time u_i. All requests are to be served within the plannig horizon H, after which vehicles return to one of the optional destination depots $f^k \in \mathcal{F}$. All nodes are connected by arcs with travel time t_{ij}, hence each vehicle chooses the destination depot f^k which is closest to the last visited location i, i.e., $f^k = \text{argmin}_{j \in \mathcal{F}} t_{i,j}$.

Vehicles are heterogeneous in terms of their loading capacity C^k, homogeneous in terms of their battery capacity Q, and characterized by initial battery levels B_0^k corresponding to their current state-of-charge (SOC). Vehicle batteries can be recharged at uncapacitated charging stations $\mathcal{S}$. It is assumed that empty vehicles can visit multiple charging stations along a route, can partially recharge, and return with minimal battery ratios r at destination depots $\mathcal{F}$. The amount of energy recharged at the charging stations in $\mathcal{S}$ is proportional to the time E_s^k spent at the facilities. In particular, the amount of energy transferred per time unit is controlled by the charging rate α_s (e.g., fast charging, slow charging). Similarly, battery consumptions $\beta_{i,j}$, with i and $j \in \mathcal{V} = \mathcal{O} \cup \mathcal{N} \cup \mathcal{S} \cup \mathcal{F}$, can be inferred from the travel times $t_{i,j}$, combined with other features by means of an energy consumption model (e.g., Pelletier et al., 2017; Goeke & Schneider, 2015).

Vehicle plans are constructed by finding minimum cost routes and observing time window, capacity, maximum ride time, and battery constraints. The cost function consists of a weighted-sum objective comprising the total vehicle travel time and user excess ride time. These terms are weighted through factors $\{c_1, c_2\}$. Five types of decisions variables are used to make decisions on the vehicle plans: (1) a binary decision variable x_{ij}^k denotes whether vehicle k sequentially visits locations i and $j \in \mathcal{V}$; (2) a continuous decision variable $T_i{}^k$ represents the time at which vehicle k begins service at location $i \in \mathcal{V}$; (3) a discrete decision variable L_i^k represents the load of vehicle k after service (i.e., units of passengers); (4) a continuous decision variable B_i^k represents the battery state of vehicle k at the beginning of service; and (5) a continuous decision variable E_s^k denotes the recharge time of vehicle k at stations $s \in \mathcal{S}$. In addition, the auxiliary decision variable R_i represents the excess ride time of request $i \in \mathcal{P}$, that is the extra time the request spends on board of the vehicles with respect to a taxi service.

Finally, the static e-ADARP can be modeled as a 3-index or 2-index mixed integer linear program (MILP). For a formal mathematical definition of the static e-ADARP, we refer the reader to Section 2 in Bongiovanni et al. (2019).

3.2. Adaptations to the static e-ADARP for dynamic operations

Static e-ADARP subproblems are encountered in the course of dynamic e-ADARP operations, i.e., at time $h \leq H$, with the arrival of new requests. These subproblems are essentially a variant of the static e-ADARP (Bongiovanni et al., 2019), with some adaptations considering the dynamic environment. Namely, in the dynamic e-ADARP new requests may be rejected, requests which were previously accepted need to be served, vehicles are on the move, and pickup/dropoff services that took place in the past cannot be modified. As a result, vehicles may initially be non-empty, part of the excess ride time characterizing planned requests may have been used, and vehicle charging phases may need to be interrupted or modified.

Service is only rejected if a new request, with pickup location $\tilde{p}$ and dropoff location $\tilde{d}$, cannot be feasibly inserted into the existing vehicle routes. This considers the constraints imposed by the e-ADARP problem. Denying service to the new request incurs a fixed cost penalty c_3, which affects the objective function through an auxiliary binary decision variable $z_{\tilde{p}}$. This cost penalty c_3 may be computed as a proportion of the total routing cost or as a high fixed cost. As such, existing vehicle plans may need to be revisited in order to find the optimal trade-off between the following components of the objective function: (1) feasibly insert the new request, (2) decrease operational cost, (3) and increase the level of service. This is obtained by re-optimizing a current static subproblem at time h, which is composed of all information related to the current vehicle locations $\bar{o}^k$, the current vehicle plans, and the new request.

With such pre-processing steps, static e-ADARP subproblems at time h can be formulated as the MILP presented in Appendix A. Note that the MILP formulation is a hard combinatorial optimization problem which cannot be solved in nearly real-time through exact solution approaches, which is important for dynamic problems. Indeed, as shown in Bongiovanni et al. (2019), only small-size instances with up to four vehicles and 24 customers can be solved through an exact solution approach in less than one minute.

4. Solution approach

This Section focuses on the data-driven two-phase metaheuristic approach which is employed to solve sequential static e-ADARP subproblems during dynamic operations. The latter are represented by an event-based simulation approach, which comprises vehicle events (i.e., departure, arrival, and charging events) and request events (i.e., generation, pickup, and delivery events). New requests $i \in \{1, \ldots, n\}$ are generated at reservation times that are constructed by subtracting in advance booking times from earliest arrival times arr_i. The booking times are computed as independent and identically distributed random variables from an exponential distribution with a rate parameter λ, which is assumed to be homogeneous across requests. Note that larger rate parameters λ induce instantaneous booking times. A detailed discussion on the event-based simulation framework is provided in Appendix B.

4.1. Two-phase metaheuristic for static e-ADARP subproblems

Consider a subproblem instance $g \in \{\text{instances}\}$, characterized by scheduled transport requests $i \in \{1, \ldots, \bar{n}\}$, with $\bar{n} < n$, and a new transport request $j = \bar{n} + 1$. The current routing solution $w_{\text{current}} \in \{\text{solutions}\}$ has a total operational cost $f(w_{\text{current}})$. As discussed in Section 3, the goal of the two-phase metaheuristic is to decrease the total operational cost by finding the right balance between the following components of the objective function: (1) feasibly insert the new transport request, along with all other inserted requests; (2) decrease the operational cost; and (3) increase the level of service. If j is feasibly inserted through a first greedy insertion phase, the objective of the second stage reduces to finding a trade-off between components (2) and (3) of the objective function. If component (1) is not feasible, then the new transport request is rejected.

4.1.1. First phase: greedy insertion algorithm

The first phase is designed to select a vehicle that can feasibly accommodate the new request j, received at time $h < H$. New requests are represented by a reservation time, i.e., generation time, and a time window around the pickup/dropoff location. Note that the time window around the dropoff location can be easily computed from the pickup location, and vice versa, by considering user maximum ride times u_j. In addition, the time windows can be narrowed down by considering the current locations and tasks of the vehicles at time h. For example, a vehicle that is picking up a request i must first finish its task before traveling from i to j. As a result, each new request has vehicle-specific earliest arrival times arr_j, which depend on the vehicle's current locations and tasks. In contrast, the latest departure times dep_j are independent of the vehicle. Vehicles that cannot arrive at the pickup location j by its latest arrival time dep_j are not candidates for the insertion.

Forward and backward slack times can be used to facilitate decisions about changes in vehicle schedules (Savelsbergh, 1992). For the e-ADARP, the maximum time interval by which a specific request i and all preceding/following nodes in the vehicle schedule can be pushed backward/forward without violating time window constraints can be computed as in Section 3.2 in Diana & Dessouky (2004). Note that, in the case of e-ADARP, service times include any charging phases. Forward slack times F_i and backward slack times $\bar{F}_i$, computed as proposed in Parragh et al. (2009), are used to identify segments of the vehicle paths that may contain the pickup and the dropoff of request j, separately. That is, having computed the forward/backward slack times for all requests i within a vehicle plan, it is possible to identify route segments where the inclusion of the pickup or the dropoff of request j would not violate time window constraints. For example, the pickup of request j must certainly be served after the last node

i within the vehicle route that satisfies the condition $T_i - \bar{F}_i < arr_j$. Similarly, the pickup of request j certainly needs to be served before the first node i within the vehicle route that satisfies the condition $T_i + F_i > dep_j$. Given capacity constraints, these segments can be further constrained. That is, knowing the maximum capacity of the vehicles C and the current vehicle loads l_i, the pickup of request j cannot be feasibly inserted when the vehicle is scheduled to travel at capacity.

The feasibility of a specific pickup-dropoff insertion option within a candidate vehicle can be further preprocessed with respect to time window and battery considerations. Time window feasibility is linearly checked by assuming that vehicles start providing service at time h and by verifying that time windows are not violated at any of the following nodes. Initial battery feasibility is evaluated by assuming that vehicles end their current idle/charging phases as soon as they have reached the minimum state of charge that allows them to directly serve the pickup and the dropoff of request j, as well as all their already scheduled requests. Note that we do not consider the insertion of new intermediate charging visits along vehicle routes. This assumption is employed to limit the computational load of the greedy insertion phase. However, new recharge/idle visits are always added at the end of each vehicle path by selecting the recharge facility that minimizes the travel time from its last visited location. Assuming that deviating vehicles to serve new requests requires a battery expense, it is sufficient to check that subtracting this expense from the vehicle initial battery states at the visited charging facilities does not return negative values. If battery constraints are violated at any of the charging facilities along a vehicle route, the vehicle is assumed to be battery infeasible. Its limited battery inventory is consequently scheduled to be replenished at a new charging visit, added at the end of its route.

After the preprocessing steps, vehicle schedules that include the new request j can be constructed by employing the scheduling algorithm proposed in Chapter 3 in Bongiovanni (2020). The goal of the scheduling algorithm is to be able to find minimal excess-time and battery-feasible schedules for fixed routes. The methodology observes that battery is an inventory that can only decrease with traveling and that it is in general beneficial to recharge as much as possible as early as possible at the visited charging stations. The remaining scheduling problem notes that user excess ride time depends on vehicle loads and that finding excess-time optimal schedules is equivalent to assigning the right amount of waiting time to the inter-station pickup/dropoff locations.

Next, maximum run time constraints can be checked by identifying requests who would experience an excess ride time increase given the insertion of the new request j. For each candidate vehicle, multiple insertions of the new request j are evaluated against the 3-term weighted objective function described in Section 3. Finally, the chosen insertion is the one that results in the lowest value of the objective function.

4.2. Second phase: machine learning-based large neighborhood search

The second phase is triggered to revisit decisions taken in the first phase as to increase the level of service and decrease the operational cost. To this aim, the

Algorithm 1: MLNS heuristic

```
Input: instance $g \in \{\text{instances}\}$; current solution $w_{\text{current}} \in \{\text{solutions}\}$; current requests
       $i \in \{1, \ldots, \bar{n}\}$; new request $j = \bar{n} + 1$, if available; LNS iterations $b = \{1, \ldots, I\}$;
       minimal neighborhood size $q_{min}$; maximal neighborhood size $q_{max}$; $m_d$ destroy
       models $d_l$; $m_r$ repair models $r_n$; classification models $\xi_{(d_l, r_n)}$ for
       $l \in \{1, \ldots, m_d\}, n \in \{1, \ldots, m_r\}$; termination condition
Output: $w_{\text{best}}$
1  Initialize: $w_{\text{best}} = w$, $w_{\text{current}} = w$, $z_i = 0\ \forall i \in \{1, \ldots, \bar{n}\}$, $z_{\bar{n}+1} = 1$ if $j \neq \emptyset$;
2  for $b \to$ 1:I do
3      Draw: $q_b \in [q_{min}, q_{max}]$;
4      Compute: $\mathbf{x}^b_{g,w,j}$ from $g$, $\bar{w}_{\text{current}}$, $j$, and $q_b$;
5      Predict: $\xi_{(d_l, r_n)} : \mathbf{x}^b_{g,w,j} \to y^b_{(d_l, r_n)}$ for $l \in \{1, \ldots, m_d\}, n \in \{1, \ldots, m_r\}$;
6      Select: $(d_{\bar{l}}, r_{\bar{n}})(b) \subseteq (d_l, r_n) \ \mid\ y^b_{(d_l, r_n)} = 1$;
7      Compute: $P_{(d_{\bar{l}}, r_{\bar{n}})}(b) \sim U(0, 1)$;
8      Draw:$(d, r)(b) = P_{(d_{\bar{l}}, r_{\bar{n}})}(b)$;
9      Compute: $w^{'}, R_b = d(b)(q_b)$;
10     if $\{i | z_i = 1\} \neq \emptyset$ then
11         $q_b = q_b + |\{i | z_i = 1\}|$;
12         $R_b = R_b \cup \{i | z_i = 1\}$;
13     Compute: $(w^{''}, z_i, j) = r(b)(w^{'}, R_b)$;
14     if $f(w^{''}) < f(w_{best})$ & $\{i | z_i = 1\} \setminus \{j\} = \emptyset$ then
15         $w_{\text{best}} = w^{'}$;
16     if accept($f(w^{''}), f(w_{current})$) is true then
17         $w_{\text{current}} = w^{'}$;
18     if termination condition met then
19         break;
```

MLNS destroys and repairs the current routing solution over multiple iterations $b \in \{1, \ldots, I\}$. At each iteration, the search uses a specific destroy-repair couple $(d, r)(b)$, which is selected among $m_d \times m_r$ competing algorithms (d_l, r_n), with $l \in \{1, \ldots, m_d\}, n \in \{1, \ldots, m_r\}$. Destroy-repair couples $(d, r)(b)$, for $b \in \{1, \ldots, I\}$, are selected according to the pseudo-code presented in Algorithm 1 and explained next.

At the beginning of iteration b, the destroy level q_b is drawn from a uniform interval bounded between a minimal neighborhood size q_{min} and a maximal neighborhood size q_{max}, which can be computed as a percentage of the number of requests $\bar{n} \leq n$ in the current routing solution w. The destroy level, instance, routing solution, and new request information (if available) forms the d-dimensional input data vector $\mathbf{x}^b_{g,w,j} = \{x^b_{g,w,j_1}, \ldots, x^b_{g,w,j_d}\}$ which is fed to destroy-repair classification models $\xi_{(d_l, r_n)}$, with $l \in \{1, \ldots, m_d\}, n \in \{1, \ldots, m_r\}$. Each classification model returns a binary output, i.e., $y^b_{(d_l, r_n)} \in \{0, 1\}$, depicting whether each destroy-repair couple (d_l, r_n) should be considered at current iteration b. The binary outputs $y^b_{(d_l, r_n)}$ indirectly account for the expected performance of each destroy-repair couple at the current iteration b. Specifically, a destroy-repair couple is only considered at iteration b if it is predicted to be among the algorithms that would bring the highest improvement in the objective function value at iteration b. Using the binary outputs $y^b_{(d_l, r_n)}$, we select a subset of the destroy-repair operators, i.e., $(d_{\bar{l}}, r_{\bar{n}})(b) \subseteq (d_l, r_n) \ \mid\ y^b_{(d_l, r_n)} = 1$. Successively, we uniformly draw the destroy-repair couple to be employed at iteration b among the selected destroy-repair couples, i.e., $(d,r)(b) = P(d_{\bar{l}}, r_{\bar{n}})(b) \sim U(0,1)$. Note that the destroy-repair operators do not necessarily improve the objective function value at each iteration b, since we consider worsening solutions during the search, either feasible or infeasible. As such, if none of the algorithms are predicted to bring any improvement in the objective function value, i.e., $y^b_{(d_l,r_n)} = 0 \ \forall \ l \in \{1,\ldots,m_d\}, n \in \{1,\ldots,m_r\}$, we can randomly select a destroy-repair $(d,r)(b)$ to be employed at iteration b or, alternatively, terminate the search.

The drawn operator $d(b)$ removes q_b requests from the current solution w_{current} at iteration b. Following the logic of the drawn operator $d(b)$, a set of requests $R_b \in \{1,\ldots,\bar{n}\}$ are removed from the current solution, which is updated to a solution $w^{'}$. Note that the vehicle schedules in $w^{'}$ do not need to be updated at this stage, since they are later re-computed through the repair operator $r(b)$. A request bank z_i is initialized to take into account the insertion of the new request j, if available, or any other request that could not be feasibly re-inserted in previous iterations, i.e., requests i for which $z_i = 1$. Consecutively, the set of requests to be re-inserted at iteration b is updated to $R_b = R_b \cup \{i|z_i = 1\}$. The request bank may be updated after the re-insertion phase through $r(b)$, as explained next. Operator $r(b)$ repairs the solution $w^{'}$ through an iterative process, which may take up to $q_b + |\{i|z_i = 1\}|$ iterations. Each pending request is independently inserted into the current vehicle plans, according to $r(b)$. All feasible re-insertions belonging to different vehicles are selected to update the vehicle plans and the request bank. If more than one re-insertion belongs to the same vehicle, the minimum cost solution can be selected. The remaining requests are re-inserted through $r(b)$ on the updated vehicle routes and up until the set of currently pending requests R_b is emptied. During the iterative process, it is possible that some of the pending requests R_b cannot be feasibly re-inserted. These requests are placed into the request bank z_i, which penalizes the objective function and is fed at the following MLNS iteration. New incumbent solutions need to contain all of the removed requests, with the exception of the new request j. That is, if $\{i|z_i = 1\} \setminus \{j\} \neq \emptyset$, the solution is infeasible. Note that during the re-insertion phase, the new request j may also be feasibly assigned to a vehicle route. In this case, j is updated to an empty set and the new request needs to be served as all the other planned requests in the vehicle routes.

Finally, a new solution $w^{''}$ is obtained by destroying the pending requests R_b from $w_{current}$ through $d(b)$ and repairing them through $r(b)$. If $w^{''}$ is feasible and reduces total cost $f(w_{best})$, the incumbent solution w_{best} is updated. Note that a simple comparison between $f(w_{\text{best}})$ and $f(w^{''})$ is not sufficient when the employed penalty cost is homogeneous between new and planned requests. For example, suppose the current solution is penalized by the existence of the new request j but it is feasible for all other planned requests. Successively, suppose that a neighborhood move has modified the vehicle plans to $w^{''}$ such that the new request j is inserted but another planned request i is placed in the bank. In this case, the solution $w^{''}$ is infeasible, although the objective function

$f(w^{''})$ may be lower than $f(w_{\text{best}})$, i.e., if serving j comes at a lower cost than serving i. The acceptance criterion, based on SA, always updates the current solution to $w^{''}$ when $f(w^{''}) < f(w_{\text{current}})$. Instead, the acceptance criterion can accept or reject worsening solutions $w^{''}$ (including infeasible solutions) through iteration-dependent probabilities. The search, initially composed of I iterations, may be prematurely terminated, for example if the maximal number of sequential non-improving iterations or a maximum time limit is exceeded.

5. Prediction problem

The prediction models employed within the MLNS are obtained through a statistical learning approach using a large dataset containing tuples $(\mathbf{x}^b_{g,w,j}, y_b)$. In this section, we define the prediction problem and depict the machine learning approximation.

5.1. Definition

Consider a problem instance g and an incumbent solution w characterized by aggregated features (i.e., input variables) $\{X^b_{g,w,j_1}, \ldots, X^b_{g,w,j_d}\}$ at iteration b. These features may correspond to scalar, boolean, or more generally categorical values (e.g., number of requests in the vehicle future plans, the presence of a new upcoming request, current vehicle tasks etc.). The d-dimensional vector $\mathbf{x}^b_{g,w,j} = \{x^b_{g,w,j_1}, \ldots, x^b_{g,w,j_d}\}$ denotes one realization of such features and defines a datapoint. Tuples $(\mathbf{x}^b_{g,w,j_e}, y^b_e)$, with $e = \{1, \ldots, N\}$, are a collection of examples of problems for which the output y^b is known, i.e., the labeled set.

Sample an arbitrary large subset of examples within the labeled set, called the training set with size N_{train}. A supervised learning approach aims at estimating a function $\xi : \mathcal{X} \to \mathcal{Y}$ which best maps the input space to the output space through the examples provided by the training dataset. If $\mathcal{Y} = \{0, 1\}$, ξ is a binary classifier. The goal of the classification problem is to minimize a loss function $\mathcal{L}$ measuring the discrepancy between the predicted and known output values from the training dataset, with size N_{train}. The generalization capabilities of the resulting model is checked on a test dataset with size $N_{\text{test}} < N_{\text{train}}$. The test dataset consists of the examples that remain from the labeled set that are not in the training set.

There exists multiple metrics providing the expected prediction error. In the proposed work, model performance is estimated by employing the following main metrics: (1) accuracy (also known as zero-one loss), (2) precision (also known as positive predictive value), (3) recall (also known as true positive rate or sensitivity), (4) specificity (also known as true negative rate). The four measures are summarized below:

$$\text{Accuracy} = \frac{1}{N_{\text{test}}} \sum_{e=1}^{N_{\text{test}}} \mathbb{1}_{[\xi(\mathbf{x}^b_{g,w,j_e})]=y^b_e} \tag{1}$$

$$\text{Precision} = \sum_{e=1}^{N_{\text{test}}} \frac{\mathbb{1}_{[\xi(\mathbf{x}^b_{g,w,j_e}), y^b_e]=1}}{\mathbb{1}_{[\xi(\mathbf{x}^b_{g,w,j_e}), y^b_e]=1} + \mathbb{1}_{[\xi(\mathbf{x}^b_{g,w,j_e})]=0 \ \& \ [y^b_e]=1}} \tag{2}$$

$$\text{Recall} = \sum_{e=1}^{N_{\text{test}}} \frac{\mathbb{1}_{[\xi(\mathbf{x}^b_{g,w,j\,e}),y^b_e]=1}}{\mathbb{1}_{[\xi(\mathbf{x}^b_{g,w,j\,e}),y^b_e]=1} + \mathbb{1}_{[\xi(\mathbf{x}^b_{g,w,j\,e})]=1\ \&\ [y^b_e]=0}} \tag{3}$$

$$\text{Specificity} = \sum_{e=1}^{N_{\text{test}}} \frac{\mathbb{1}_{[\xi(\mathbf{x}^b_{g,w,j\,e}),y^b_e]=0}}{\mathbb{1}_{[\xi(\mathbf{x}^b_{g,w,j\,e}),y^b_e]=0} + \mathbb{1}_{[\xi(\mathbf{x}^b_{g,w,j\,e})]=0\ \&\ [y^b_e]=1}} \tag{4}$$

Here, the indicator function $\mathbb{1}_{[Z]=a}$ equals 1 if $[Z] = a$, and 0 otherwise. Note that the trade-off between precision and recall can be measured through the F1 score, which is defined as follows:

$$\text{F1} = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}} \tag{5}$$

For a discussion exploring the meaning and differences between the proposed validation measures, the reader is referred to Section 9.2.4 in Hastie et al. (2009).

5.2. Machine learning approximation

In this work, the prediction problem is tackled through ensemble learning, namely random forest (RF) classification (Breiman, 2001). The goal of ensemble learning is to combine simple and fast learners to obtain better performance. In the case of RF, the weak learners are classification and regression trees (CARTs, Breiman, 2017). CARTs recursively partition the input space through a series of binary splits chosen through a mathematical model which maximizes a goodness of split function. The depth of the tree is typically controlled by hyper-parameters which provide an upper bound on the maximal number of splits and, consequently, sub-regions. An indication of the values to be adopted for such hyper-parameters may be found in the literature (Liaw et al., 2002). Appropriate hyper-parameter values can be optimized for the treated problem through a search mechanism (e.g., grid search, random search) and k-fold cross validation (Bergstra & Bengio, 2012).

For a classification problem, the value, i.e., class, of each sub-region is decided by computing the largest number of representative samples in the sub-region (the "majority vote"). The predicted value for a new input point is then obtained by passing the point through the tree until a final node (or sub-region) is reached. In order to increase the precision of the prediction, multiple decision trees are typically combined through a technique called *bagging*, which essentially generates multiple models based on bootstrap samples of the input space (Breiman, 1996). Random forests attempt to further increase the performance of the predicted models and de-correlate trees by choosing a subset of the feature space at every split in the tree. The final prediction is the aggregation of the predictions of all models. For a classification task, the aggregation corresponds to the most frequently-predicted class. For random forests, as for any other ML methodology using bagging, the performance of the predicted models can be also measured by the out of bag (OOB) error, that is the average prediction error from each training datapoint using only the trees that did not contain it in their bootstrap sample.

Finally, random forests provide an intuitive way to interpret the average feature importance, measured by the average impurity decrease. Impurity represents how well the trees split the data and can be measured by means of the Gini index (Lerman & Yitzhaki, 1984). The impurity decrease measures the reduction of impurity between successive nodes in the trees. Then, the feature importance within a tree is interpreted as the total impurity decrease brought by each feature within the tree. For a forest, the total impurity is averaged across all trees.

6. Labeled dataset generation

The labeled dataset is extracted through the event-based simulation approach described in Section 4 and detailed in Appendix B, on several instances of the dynamic e-ADARP. The information needed to produce the labels is only retrieved during the second phase, i.e., MLNS, according to the procedure explained next. Consider an initial subproblem i, consisting of a routing solution w_{current} and cost $f(w_{\text{current}})$. The initial subproblem may be modified by applying any of the available destroy-repair operators (d_l, r_n) with $l \in \{1, \ldots, m_d\}, n \in \{1, \ldots, m_r\}$. The outcome y^b of each destroy-repair couple is stochastic, since it depends on the randomly-drawn destruction level q_b, i.e., the neighborhood size, and on diversification hyperparameters. The latter are designed to slighly perturb the logic of the destroy-repair operators and diversify the search (see Ropke & Pisinger, 2006).

In order to gather meaningful statistics, each destroy-repair operator (d_l, r_n) is applied to w_{current} for M times considering different values of q_b. Namely, the destruction level removes a minimal number of requests (e.g., four requests) and may destroy up to 15 percent, 25 percent, 35 percent, and 45 percent of the vehicle routes during the M applications of (d_l, r_n). Ideally, M should be large enough in the attempt to de-randomize the statistics associated with the performance of each operator. The objective function improvement between the routing solution before the move (i.e., w_{current}) and the routing solution after the move (i.e., $w^{'}$) from all of the M trials is stored. Objective improvement distributions on the given subproblem are determined at the end of the M trials for each destroy-repair couple (d_l, r_n). An example of such distributions is given in Figure 1.

Several statistical measures can be retrieved to characterize the expected objective improvement from the algorithm distributions (e.g., mean, median, mode, percentiles). The choice of the appropriate statistics depends on the goal of the prediction problem. The considered goal is to be able to predict the algorithm that most likely moves the current solution in a positive direction for the objective function. As such, the labeling considers the percentage of accepted neighborhood moves, which can be computed through a simulated annealing (SA) approach. Specifically, given the percentage of accepted neighborhood moves from each destroy-repair operator during the M trials, we provide positive

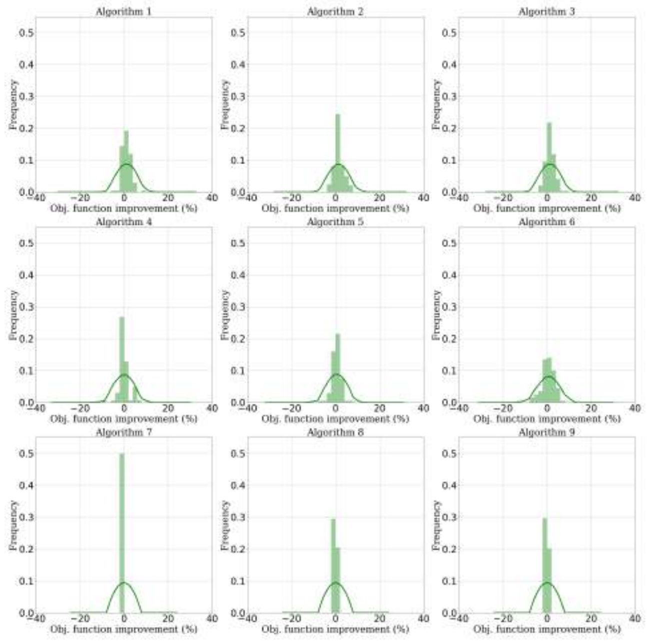


Figure 1: Example of algorithm performance distributions at iteration b and after M trials

labels to all algorithms that performed the best. That is, the algorithms which did not deviate by more than ten percent with respect to the algorithm resulting in the highest percentage of accepted neighborhood moves. Note that negative improvements may be either accepted or rejected by the SA approach at any iteration in the search. Specifically, highly infeasible solutions are prevented by the SA approach, namely its temperature and cool rate parameters.

As in the MLNS approach, the destroy-repair operator $(d, r)(b)$ is consecutively uniformly drawn among the algorithms which were labeled positively. Finally, the next move is computed by re-applying the drawn destroy-repair operator $(d, r)(b)$ on the current solution at iteration b. This last procedure is timed and the search is exited when it exceeds a time limit of five seconds. Note that this time limit has been set in consideration of the computational burden of the proposed labeling, as there are as many neighborhood moves as the number of iterations, destroy-repair operators, and trials per destroy-repair couple.

7. Feature selection

The encountered subproblems at iteration b are characterized by several aggregated features $\mathbf{x}^b_{g,w,j}$ which are extracted from the instance and routing solution at hand. As noted in Kerschke et al. (2019), the extracted features should be informative (i.e., relevant for distinguishing between problem instances),

Table 1: Features description

Size and spatial distribution of requests	**Routing solution**
1. Problem size	25. Used vehicles [%]
2. Planning horizon elapsed [%]	26.-27. Number of requests per vehicle (avg., std.)
3.-4. Radius (avg., std.)	28. Vehicle travel time (avg.)
5. Area covered by vehicles [%]	29. User excess ride time (avg.)
6.-7. Pickups/dropoffs in convex hull of dropoffs/pickups [%]	30.-31. Vehicle loads (avg., std.)
Upcoming request to be inserted	32.-33. Vehicle tour lengths (avg., std.)
8. New request to insert (binary)	34.-35. Number of edges (avg., std.)
9.-10. Distance between vehicle locations and pickup of new request (avg., std.)	36.-37. Charging time per visited charging station (avg., std.)
11. Dropoff of new request in the convex hull of the planned requests (binary)	38.-39. Travel time per traversed arc (avg., std.)
12.-15. Distance between new pickup/dropoff and planned requests (avg., std.)	40. Unique travel time arcs [%]
Vehicle current location and states	**Current and past LNS iterations**
16. Vehicles in convex hull of planned requests [%]	41. Neighborhood size [%]
17.-18. Distance vehicles from centroid of area (avg., std.)	42. Cost difference between current solution and initial solution [%]
19. Intra-vehicle distance (avg.)	43. Cost difference between current solution and last incumbent [%]
20. Empty vehicles [%]	44. Requests in request bank [%]
21.-22. Current vehicle loads (avg., std.)	45. Decrease in temperature from initial SA temperature [%]
23.- 24. Vehicle state of charge (avg., std.)	46. Iteration number

interpretable (i.e., able to provide insight into instance properties), cheaply computable (i.e., retrievable within milliseconds), generally applicable (i.e., applicable to a broad range of problem instances), and complementary (i.e., un-correlated). Note that the performance of the machine learning approximation specifically adopted in this study, i.e., RF, is not particularly susceptible to correlated input variables. However, correlated variables may affect the interpretability of RF, notably its embedded feature importance evaluation. In the case of vehicle routing problems, most extracted features are necessarily inter-correlated since they are all connected through decision variables and constraints in the problem definition. As such, for vehicle routing problems, feature importance needs to be interpreted in consideration of all possible correlations between the several proposed features. In the dynamic e-ADARP, such aggregated features relate to: (1) the size and spatial distribution of the requests, (2) the upcoming request to be inserted, (3) the vehicle current states and spatial distribution, (4) the routing solution, (4) current and past LNS iterations. Overall, we determine 46 features, grouped in the five categories summarized in Table 1 and explained next.

7.1. Size and spatial distribution of requests

We are interested in exploiting features that relate to the demand distribution, and therefore do not depend on the specific routing solution encountered through the MLNS. Specifically, we retrieve the total number of requests assigned to the vehicle routes (i.e., 1. the problem size), which were received from the beginning of the planning horizon (2.). Then, we extract the radius (3.-4.), that is the average/standard deviation distance between the planned requests and the centroid defined by the same (Gomes & Selman, 2001). The ratio is a useful metric in detecting whether requests are more or less homogeneously distributed among them. We continue by extracting the ratio between the area of the circle containing all of the assigned requests to the total area covered by the ridesharing service (5.). This ratio is useful in determining whether the assigned requests are concentrated or scattered within the service zone. Finally, we compute the percentage of pickups/dropoffs contained in the convex hull determined by the dropoff/pickup locations within the vehicle routes (6.-7.).

This last feature is helpful in determining whether the instance is characterized by clustered or homogeneously distributed pickup and dropoff locations.

7.2. Upcoming request to be inserted

A given subproblem at iteration b may be characterized by the existence of an upcoming request which could not be feasibly inserted into the vehicle plans through the first phase of the two-phase metaheuristic approach (Section 4.2). The existence of a new un-inserted request highly penalizes the objective function, by the cost parameter c_3. Therefore, we start by extracting a binary variable determining whether the subproblem considers a new request to be inserted (8.). If a new request exists, we characterize the location of the upcoming request by three distinct features. First, we determine the average distance between the vehicle current locations and the pickup of the new request, and its standard deviation (9.-10.). Second, we determine whether the dropoff of the new request lies within the convex hull defined by all requests in the vehicle plans (11.). That is, whether the new request follows the demand pattern defined by the planned requests. Third, we determine the average distance between the pickup and dropoff of the new request to all of the requests in the vehicle plans, and its standard deviation (12.-15.). Specifically, features (11.-15.) help quantify the deviation that may be needed to serve the new request.

7.3. Vehicle current states and spatial distribution

We are interested in retrieving the vehicle current states and spatial distribution at the time the subproblem is re-optimized as this affects the feasibility of neighborhood moves. With respect to the vehicle spatial distribution, we first retrieve the percentage of vehicles contained in the convex hull defined by all of the planned requests (16.). The convex hull defines a perimeter of service that is dynamically adjusted in the course of operations. Second, we extract the average distance between the vehicle current locations and the centroid of the area covered by the ridesharing service, as well as its standard deviation (17.-18.). This feature helps understand the current spatial distribution of the fleet in the service area. Third, we retrieve the average intra-vehicle distance (19.). Differently from the previous feature, intra-vehicle distance helps distinguish cases in which vehicles are homogeneously dispersed in the service area from cases in which groups of vehicles are clustered together. With respect to the vehicle current states, we retrieve the percentage of currently empty vehicles (20.), the average current load of the vehicles and its standard deviation (21.-22.), the average vehicle SOC and its standard deviation (23.-24.).

7.4. Routing solution

The iterative solution approach for the e-ADARP explores many solutions that do not differ in terms of the demand distribution and vehicle current states but do differ in terms of route planning. As such, we are particularly interested in extracting valuable information to differentiate between routing solutions. To this aim, we retrieve information on the percentage of utilized vehicles in the solution (25.), the average number of planned requests per vehicle and its standard deviation (26.-27.), the average vehicle travel time (28.) and excess ride time per served request (29.). Furthermore, we extract aggregate information derived from the sequence in which requests are served within the vehicle plans. Specifically, we retrieve the average vehicle loads along their planned routes and their standard deviation (30.-31.), the average vehicle tour length and its standard deviation (32.-33.), the average number of edges per vehicle tour and its standard deviation (34.-35.), and the average vehicle charging time per visited station and its standard deviation (36.-37.). We further explore features which depend on the vehicle routes and are derived from the travel time cost matrix. Specifically, we extract the average vehicle travel time per traversed arc and its standard deviation (38.-39.), the fraction of unique travel times from the vehicle plans (40.). These last features help determine whether vehicle routes are composed of edges characterized by similar/equal distance (i.e., travel time).

7.5. Current and past LNS iterations

At any given iteration, it is possible to exploit information from previous iterations of the search. That is, we would like to know the size of the neighborhood that we are currently willing to explore (41.) but also how far we are located, in terms of objective function value, from the beginning of the search. To this end, we retrieve the percentage cost difference between the current solution and the initial solution (42.), the percentage cost difference between the current solution and the last incumbent solution (43.), the percentage of requests added to the request bank (44.), the percentage decrease in temperature for the initial SA temperature (45.), and the iteration number (46.).

8. Numerical experiments

This work employs real data from 24,400 Uber ride-shares during a one-week period in San Francisco (CA, USA), obtained by extracting pickup/dropoff locations and times from published GPS logs[1], as described in Bongiovanni et al. (2019). The time series showing the number of ridesharing requests from the Uber dataset is provided in Figure 2. For the purpose of our tests, we slice the dataset into 244 100-request dynamic scenarios. Note that larger scenarios are not necessarily more difficult to solve, as complexity depends on the size of the encountered subproblems, which in turn depends on the demand distribution and arrival rate. We construct a 15-minutes time window around the request pickup/dropoff times and assume instantaneous booking times (i.e., $\lambda \to \infty$). The travel time between latitude/longitude locations is approximated by the Eucledian distance multiplied by a factor of 300. The maximum ride time is considered fixed for all requests and is set to 30 minutes. We consider a fixed fleet size of 10 vehicles, with a maximal capacity of 15 passengers, a

[1] `https://github.com/dima42/uber\protect\discretionary{\char\hyphenchar\font}{}{}gps-analysis/tree/master/gpsdata`

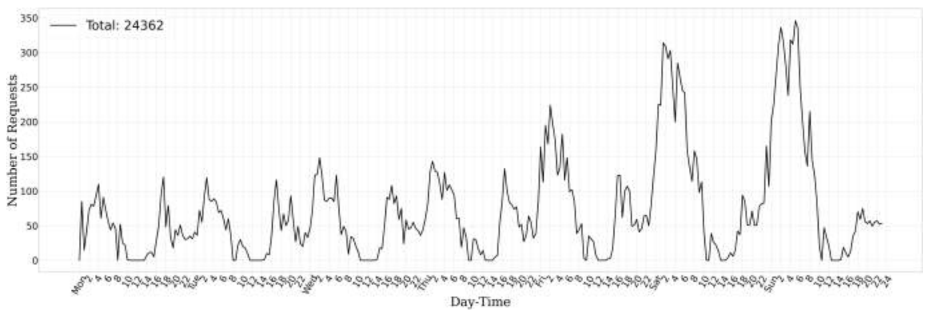


Figure 2: Number of ridesharing requests from the Uber dataset

nominal battery capacity of 14.85 kWh, and discharge/recharge rates of 0.055 kWh per minute. These values are extracted from the technical specifications of the electric-autonomous shuttles manufacturer Navya[2]. The fleet size has been chosen such that, under the fore-mentioned conditions, 65 percent of the total demand can be served by a greedy insertion algorithm, on average. As in Bongiovanni et al. (2019), we extract the locations of the charging stations form the Alternative Fueling Station Locator from the Alternative Fuels Data Center (AFDC) of the U.S. Department of Energy[3]. We assume that vehicles can recharge along their routes to a set of 10 randomly-chosen charging stations, which also serve as destination depots. Vehicles can return to destination depots with no less than 10 percent of remaining battery capacity, i.e., $r = 0.1$.

The simulation-based optimization procedure described in Section 4 was implemented with Julia v. 1.2 (Dunning et al., 2017) and applied to each one of the 100-request dynamic scenarios. Results were obtained by running each of the simulated scenarios in parallel, by using an Intel Xeon based cluster employing two cores per node on Skylake processors running at 2.3 GHz. The second re-optimization phase is triggered any time an incoming request cannot be directly inserted and when there are at least four planned requests in the vehicle routes. Solutions that do not include the incoming request or, succesively, any other request from the vehicle routes incur into an arbitrararily-high fixed penalty cost $c_3 = 1000$. As in Bongiovanni et al. (2019), c_1 is set to 0.75 while c_2 is set to 0.25, i.e., the total vehicle travel time accounts for 75 percent of the objective function score and the total excess ride time accounts for 25 percent.

The destroy and repair operators considered in this work closely follow the ones proposed in Ropke & Pisinger (2006). Namely, we adopt all of their destroy (i.e., random, Shaw, and worst) and most of their repair (i.e., basic greedy, regret-2, and regret-3) heuristics. As such, the destroy-repair couples considered in this work, i.e., (d_l, r_n), are obtained by combinations of $d_l \in \{1 : \text{Random (R)}, 2 : \text{Shaw (S)}, 3 : \text{Worst (W)}\}$ and $r_n \in \{1 : \text{Greedy (G)}, 2 : \text{Regret2 (R2)}, 3 :$

[2]`https://www.hevs.ch/media/document/1/fiche-technique-navettes-autonomes.pdf`
[3]`https://afdc.energy.gov/stations/#/find/nearest`

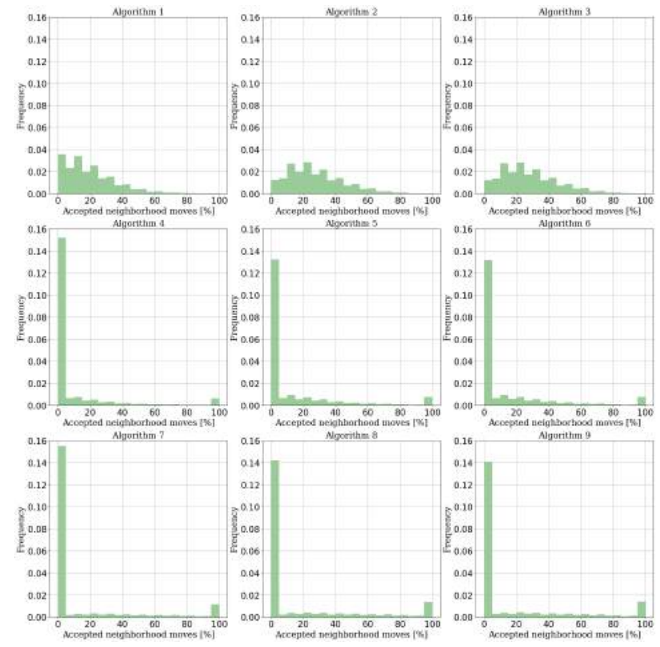
Regret3 (R3)}. Namely, we consider the following LNS algorithms (d_l, r_n): $\{\text{Alg. } 1 : \text{R-G}, \text{Alg. } 2 : \text{R-R2}, \text{Alg. } 3 : \text{R-R3}, \text{Alg. } 4 : \text{S-G}, \text{Alg. } 5 : \text{S-R2}, \text{Alg. } 6 : \text{S-R3}, \text{Alg. } 7 : \text{W-G}, \text{Alg. } 8 : \text{W-R2}, \text{Alg. } 9 : \text{W-R3}\}$. The search is composed of a total of 100 iterations and is notably shorter than what is typically reported in the literature. This choice is motivated by the limited computational time available to make modifications to the vehicle plans in the context of online operations (e.g., 5 seconds). Consequently, the search does not have time to explore the solution space for thousands of iterations.

As in Ropke & Pisinger (2006), the SA parameters, notably the initial temperature and the cooling rate, are set by assuming that a move that is 5 percent worse than the current solution is accepted with a 50 percent probability and that this probability converges to zero towards the end of the search. Note that given that infeasible solutions incur the high cost parameter c_3, they are mainly allowed to be explored at the beginning of the search. That is, if the incoming request can only be inserted at the cost of infeasibility for another request. In this case, the search begins from an infeasible solution and attempts to recover feasibility in successive iterations. This choice is motivated by the limited time to modify vehicle routes, in the context of our online approach. Note that preliminary results had confirmed that over-exploring infeasible solutions in the context of real-time decision making may decrease our ability to find locally optimal solutions.

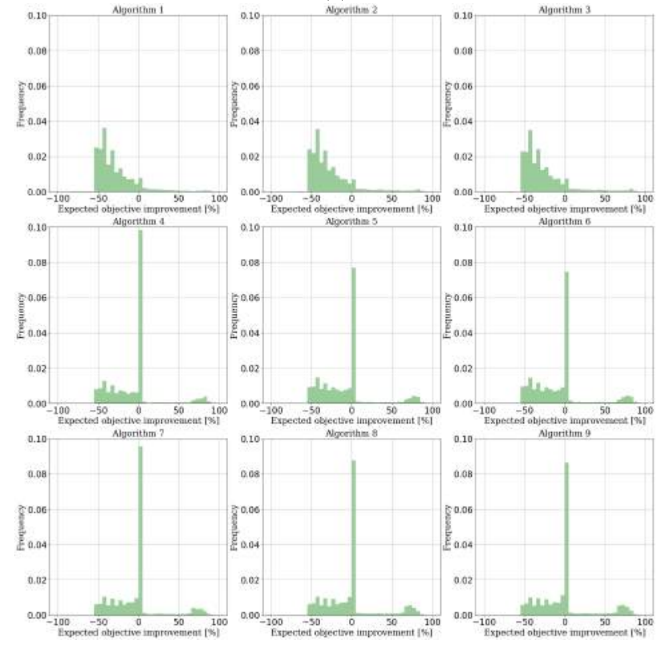
8.1. Labeled dataset

The labeled dataset is constructed following the principle described in Section 6. At each iteration during the search, we employ each of the nine competing destruction-repair algorithms a sufficiently large number of times to collect meaningful statistics, i.e., $M = 200$ times. The size of the neighborhood is drawn from a uniform interval which considers removing and reinserting at least four requests (i.e., $q_{min} = 4$) and up to 45 percent (i.e., $q_{max} = 0.45 \times \bar{n}$) of the vehicle routes. For each algorithm, the expected performance is successively computed as the ratio of accepted neighborhood moves over the M trials.

This procedure resulted in a total of 1,728,936 labeled datapoints, homogeneously distributed among the nine competing algorithms. Distributions on the ratio of accepted neighborhood moves from the labeled dataset are shown in Figure 3(a). As expected, the first three LNS algorithms, which refer to the random removal operator, feature distributions that resemble the normal distribution. The remaining six LNS algorithms feature a peak at low acceptance rates below five percent and a second peak at high acceptance rates above 95 percent. This result is to be expected since these LNS algorithms employ destruction operators that follow a specific logic that may work remarkably well or poorly on selected problems. This last observation is further confirmed by the distributions on the expected objective function improvement for accepted neighborhood moves exceeding 50 percent. As shown in Figure 3(b), the first three LNS algorithms are more likely to produce neighborhood moves of lower quality with respect to the remaining six LNS algorithms. Furthermore, these LNS algorithms seem to be able to both refine and highly improve the current solution.

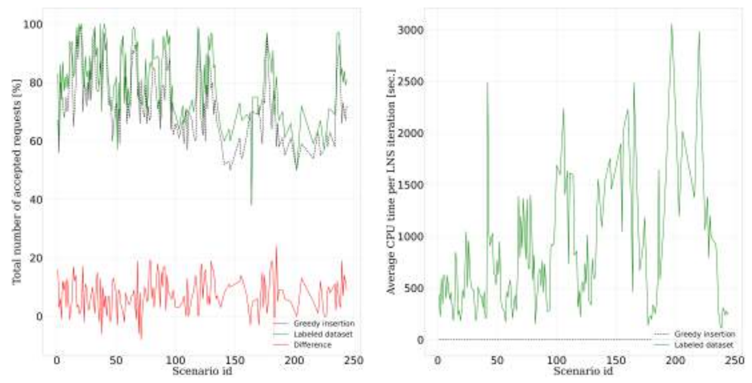


(a)



(b)

Figure 3: Labeled dataset: (a) percentage of accepted neighborhood moves, (b) expected objective function improvement with at least 50 percent accepted neighborhood moves



(a) (b)

Figure 4: Labeled dataset and simple insertion: (a) Difference in terms of total number of accepted requests, (b) Average CPU time at each iteration of the search

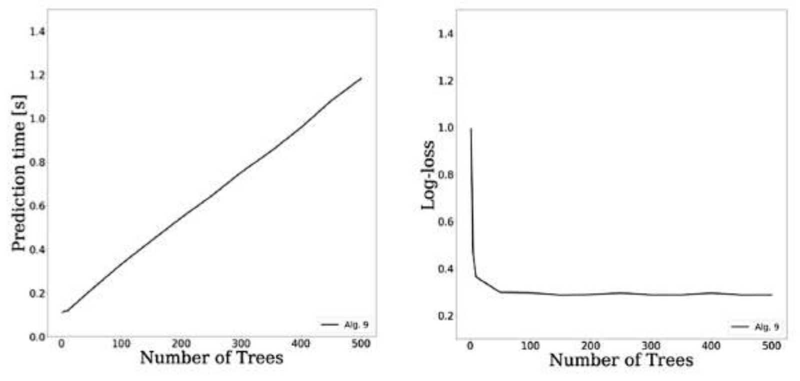
The produced labeled dataset in Figure 3 contains examples of moves from the re-optimization policy we aim at learning. Specifically, our re-optimization policy is applied to static subproblems, encoutered in the course of the dynamic operations, to increase our chances of inserting new requests. Figure 4(a) shows the difference in the total number of accepted requests between the labeled dataset and a simple greedy insertion strategy (i.e., the first phase in the two-phase heuristic approach). As it can be noted, the proposed re-optimization policy has a positive effect on the total number of accepted requests, which increases by up to 24 percent with respect to a simple greedy insertion policy in the simulated dynamic scenarios. In only about eight percent of the scenarios, the re-optimization phase may lead to a lower number of accepted requests, by about three percent on average.

As shown in Figure 4(b), the extensive methodology used to produce the labeled dataset is not suitable for real-time decision processes given its computational burden. Specifically, each iteration of the extensive LNS-based approach used to produce the labeled dataset can take up to about 50 minutes. As such, the re-optimization policy is learned through a statistical learning approach and efficiently re-applied to new online problem instances.

8.2. Machine learning results

A total of nine random forest classifiers are trained on the labeled dataset by the use of the ScikitLearn library[4]. Information from a 20 randomly-selected dynamic scenarios is completely excluded from the labeled dataset to compose a validation set for the re-application of the trained models online. As customary in the ML literature, 75 percent of the remaining labeled set is selected for training, and the other 25 percent for testing. The dataset is split into training

[4]`https://scikit-learn.org/stable/`



(a) (b)

Figure 5: Relationship between the size of the random forest and (a) the prediction time, (b) the log-loss prediction accuracy for RF classifier 9.

Table 2: Performance measures obtained for the nine RF regressors on the train and test dataset

RF classifier	Imbalance 0/1	Accuracy	Precision	Recall	Specificity	OOB
1	1.97	0.80	0.70	0.73	0.84	0.80
2	0.75	0.75	0.78	0.78	0.71	0.86
3	0.72	0.75	0.78	0.79	0.70	0.75
4	9.60	0.94	0.68	0.76	0.96	0.94
5	5.07	0.89	0.64	0.77	0.91	0.89
6	4.99	0.88	0.63	0.78	0.91	0.88
7	6.50	0.90	0.61	0.75	0.93	0.91
8	4.48	0.87	0.62	0.72	0.90	0.87
9	4.41	0.86	0.62	0.71	0.90	0.87

and testing such that the two datasets feature an equal ratio between the 0-1 labels.

The size of each RF classifier is determined experimentally by inspecting the predictive performance of the classifiers for forests composed of 1 to 500 trees. As shown in Figure 5(a), prediction time is quasi-linearly correlated with the size of the forest. Specifically, increasing the size of the forest from 1 to 500 trees increases the prediction time by a factor of 10. However, as shown in Figure 5(b), in our case the RF classifiers do not need to be composed of more than 100 trees to keep confident predictions. As such, we only train larger forests composed of 500 trees to extract feature importance statistics. On the training dataset, each RF classifier is further optimized with respect to selected hyperparameters, i.e., the maximal number of features and minimal number of samples per split, using grid search and k-fold cross-validation. The optimal parameters are chosen from the results obtained on five folds and by considering the F1 score.

Table 2 shows performance measures obtained for each of the nine trained classification models by applying them on the test dataset. Column “Imbalance 0/1” shows the ratio between class 0, i.e., do not employ RF classifier i, and class 1, i.e., do employ RF classifier i. As it can be noticed, some of the RF classifiers are trained on highly imbalanced examples. Specifically, the training dataset

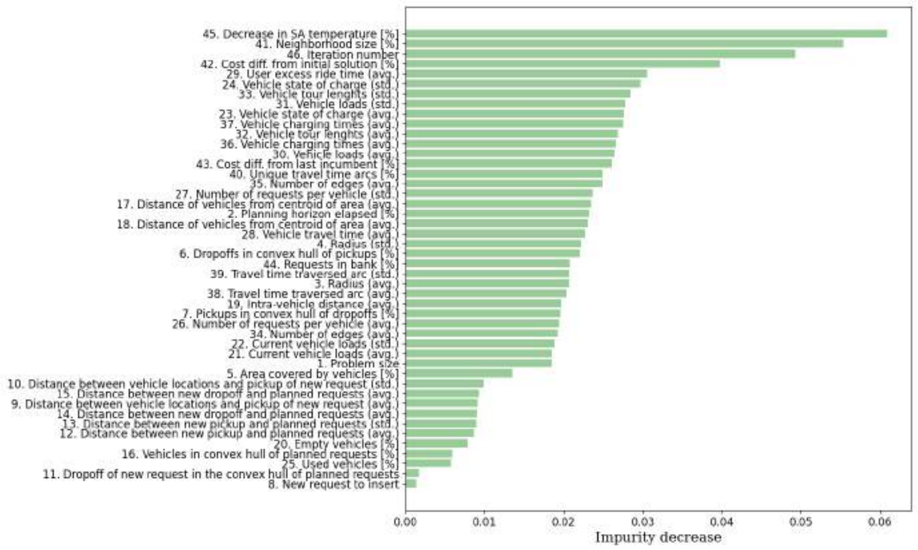


Figure 6: Average feature importance

contains more examples of class 0 rather than class 1, with the exception of RF classifiers 2 and 3. Imbalanced datasets are tackled by assigning weights to the classes, which are computed based on the bootstrap sample for every tree grown (Chen et al., 2004). However, this feature of the datasets may be responsible for the relatively lower precision and recall metrics. Nonetheless, as shown by the recall (i.e., the percentage of positive labels classified as such) and the specificity (i.e., the percentage of negative labels classified as such), the RF classifiers can accurately predict both class 0 and class 1. Note that in the case of imbalanced data the accuracy is not very meaningful, we therefore choose it as our metric of preference. Finally, note that RF classifiers 1, 2, and 3, which are associated with the random destroy operator, show relatively lower performance metrics.

8.3. Feature importance

The average importance of the 46 extracted features, described in Section 7, for the classification problem is shown in Figure 6. As a reminder from Section 5.2, in RFs feature importance is measured by the average decrease in impurity brought by each feature and between successive nodes in each tree. Impurity can be interpreted as the average variance reduction brought by each feature. As it can be noticed from Figure 6, the first four features decrease the average impurity the most. These features relate to the current SA temperature (or the correlated current LNS iteration), the drawn neighborhood size, and the current objective function improvement with respect to the beginning of the search. Indeed, our ability to improve the current solution depends on history from previous iterations and on the size of the explored neighborhood. For example, it is likely that a solution is highly improved at the beginning of the search and only refined at consecutive iterations. Moreover, note that the SA temperature decreases with following iterations. This implies that, at later iterations, accepted solutions can only strictly improve the current objective function. It is therefore reasonable to assume that worsening solutions are most probably accepted at the beginning of the search, as they are controlled by the SA temperature. Solution acceptance depends on the current objective function value, which is provided by the fourth most important feature with respect to the objective function value at the beginning of the search. The next three most important features relate to user excess ride times, vehicle state of charge, and vehicle tour lenghts. Indeed, these features play a significant role in the e-ADARP and impose hard timing constraints. Finally, note that there are features which do not seem to play a significant role in the studied problem. Some of these features relate to the number of employed and idle vehicles. Note that, in the studied scenarios, vehicles are over-capacitated and so capacity does not pose an issue for feasibility. This feature may therefore impact the importance of vehicle use and availability. Finally, it is also interesting to note that the presence of new requests to be inserted does not seem to play a significant role in the ability of the algorithms to improve the current solution. This last observation can be interpreted in two ways. First, all algorithms may find new insertions difficult. Second, all algorithms may find new insertions easy, as new requests can always be inserted at the cost of infeasibility at the beginnig of the search.

8.4. MLNS Results

The trained RF classifiers are re-applied on the encoutered static e-ADARP subproblems on the dynamic e-ADARP instances from the validation dataset, in the context of the MLNS explained in Section 4.2. The proposed MLNS methodology is compared to a large neighborhood search (LNS), a random selection mechanism (RAND), and an adaptive large neighborhood search (ALNS). The LNS approach is composed of a fixed destroy-repair couple, namely a worst removal operator and a greedy reinsertion operator. The RAND approach randomly selects destroy-repair couples at each iteration during the search. The ALNS approach follows the score updating methodology proposed Ropke & Pisinger (2006). However, given the limited number of iterations to gather statistics, the parameter of the ALNS are set such that operators weights are updated more frequently and react faster to changes in the effectiveness of the operators. Specifically, we set the ALNS reaction factor r to 0.4 and update the operators scores every 10 iterations.

We assume that vehicle route modifications are performed on processors with eight cores and that computations do not need more than two cores. As such, each search mechanism (i.e., LNS, RAND, ALNS, and MLNS) is run on four parallel optimization processes which terminate after 100 iterations. At the end of each parallel re-optimization, we select the solution resulting in the maximal objective function value improvement for the current static e-ADARP subproblem.

In order to compare the outcomes of LNS, RAND, ALNS, and MLNS, we simulate each dynamic instance from the validation set 150 times, resulting in

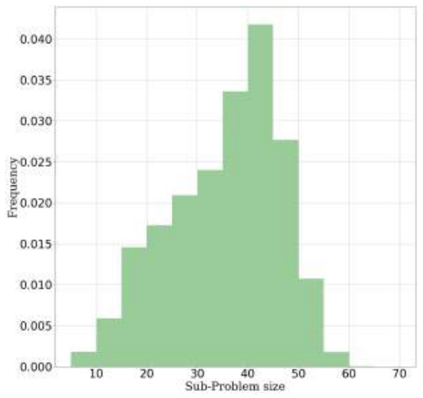


Figure 7: Distribution of the size of the encountered e-ADARP subproblems

Table 3: Percentage of times LNS, RAND, ALNS, MLNS produces the highest-quality solution, based on the subproblem size and within a limit of 100 iterations

Size	**#Subproblems**	**NONE** [%]	**LNS** [%]	**RAND** [%]	**ALNS** [%]	**MLNS** [%]
5	25	28.0	8.0	24.0	32.0	8.0
10	1036	35.33	19.50	8.59	17.95	18.63
15	3264	37.53	15.56	2.91	17.06	26.93
20	7152	34.06	16.47	2.04	18.55	28.87
25	8132	40.54	15.43	1.80	15.58	26.65
30	9497	35.74	15.33	1.74	17.06	30.14
35	11505	34.69	15.15	1.89	16.04	32.23
40	16177	45.14	12.01	2.26	12.78	27.81
45	18154	54.65	10.40	2.16	10.25	22.54
50	10568	60.72	8.63	2.02	8.39	20.23
55	3729	66.24	8.50	1.45	7.62	16.20
60	470	65.53	7.87	3.19	7.66	15.74

about 90,000 static e-ADARP subproblems. We then retrieve performance statistics on the competing search mechanisms on each static e-ADARP subproblem encountered while running the simulated dynamic instances. The distribution on the size of the encountered static e-ADARP subproblems is provided in Figure 7. Note that in the course of the dynamic e-ADARP scenarios, we solve subproblems of size ranging between 5 and 60 requests, and most frequently solve subproblems of size 40-45 requests.

Table 3 shows the percentage of times that LNS, RAND, ALNS, and MLNS produced the highest-quality solution, based on the objective function value and within a limit of 100 iterations. The results are organized by increasing size of the encountered static e-ADARP subproblems. Note that the frequency on the size of such subproblems follows the distribution shown in Figure 7. Column "none" represents the percentage of times that none of the search mechanisms could improve the solution of the static e-ADARP subproblem at hand. Note that the size of the encountered subproblem linearly relate to our ability to improve the routing solution. In particular, for subproblems of size 45 and above, we are able to improve the current routing solutions less than 50 percent of the times. However, these are also the subproblems in which we note the advantage of employing MLNS with respect to the other search mechanisms. Note that

Table 4: Expected difference in the objective function improvement based on the subproblem size and within a limit of 100 iterations: MLNS - LNS, MLNS - RAND, MLNS - ALNS

Size	#Subproblems	MLNS-LNS [%]	MLNS-RAND [%]	MLNS-ALNS [%]
5	18	4.25	-0.01	-0.01
10	670	2.96	0.38	0.85
15	2039	22.28	4.37	4.48
20	4716	28.24	6.36	5.11
25	4835	21.57	6.04	6.15
30	6103	25.99	6.81	6.53
35	7514	27.32	9.28	8.35
40	8875	24.34	9.52	9.25
45	8232	19.25	7.97	7.89
50	4151	14.88	7.46	7.06
55	1259	8.33	2.64	1.60
60	162	-1.45	-1.17	-0.73

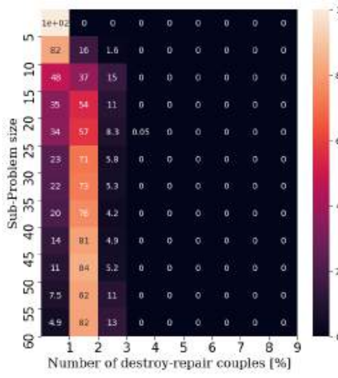
Table 5: Percentage of times LNS, RAND, ALNS, MLNS insert new requests, based on the subproblem size and within a limit of 100 iterations

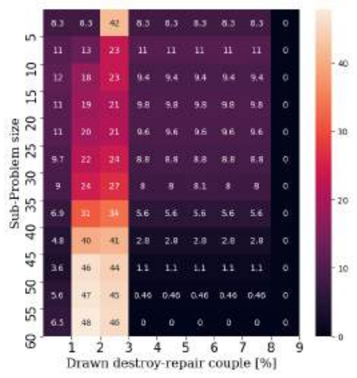
Problem size	#Sub-problems	LNS [%]	RAND [%]	ALNS [%]	MLLNS [%]
5	18	88.89	94.44	94.44	94.44
10	670	3.73	7.16	6.57	7.91
15	2039	14.96	39.33	39.28	45.46
20	4716	25.91	55.03	56.79	63.80
25	4835	36.96	57.68	57.5	65.92
30	6103	33.62	58.59	58.95	67.61
35	7514	28.13	51.96	53.21	64.19
40	8875	22.54	42.86	43.30	55.74
45	8232	18.29	34.16	34.27	45.15
50	4151	15.15	25.87	26.43	36.40
55	1259	10.17	18.67	20.17	22.32
60	162	15.43	14.81	14.20	12.96

MLNS is only comparable to the other search mechanisms for small-size problems of size 10 or less, as these problems can be easily solved by all approaches.

Table 4 shows the expected difference in the solution quality between LNS, RAND, ALNS, and MLNS, based on the percentage improvement in the objective function value and within a limit of 100 iterations. As for the previous table, the results are organized by increasing size of the encountered static e-ADARP subproblems. As it can be noted, MLNS outperforms LNS by up to about 28 percent on instances of size 15 to 50, on average. On the same instances, it also outperforms RAND by up to about 9.5 percent and ALNS by up to about 9.25 percent, on average. Only for small-scale and large-scale problems, MLNS is outperformed by the competing search mechanisms. This can be explained by the fact that MLNS is trained on few examples of subproblems of such size and it can therefore hardly generalize to small-scale and large-scale static e-ADARP subproblems. Furthermore, there is little room for improvement for small subproblems, e.g., with 0-5 requests, and for large subproblems, with 55-60 requests. In the former case, the limited size of the subproblems implies a limited number of options for destroying and repairing vehicle paths, and thus improving the objective function. In the second case, the large size of the subproblems implies a limited number of feasible options to destroy and repair the crowded vehicle paths.

Table 5 shows the percentage of times that LNS, RAND, ALNS, and MLNS inserted new requests within a limit of 100 iterations. These new requests cannot





(a)

(b)

Figure 8: Heatmap of the percentage of times: (a) we draw between 1 to 9 algorithms, (b) algorithm 1 to 9 was drawn.

not simply be inserted within the vehicle plans via a greedy insertion algorithm (i.e., the first phase of our two-phase approach). As with the previous tables, the results are organized by increasing size of the static e-ADARP subproblems encountered. As it can be noted, MLNS is able to insert a higher percentage of new requests for subproblems between 10 and 55 requests. Specifically for subproblems with a size between 30 and 50 requests, MLNS increases the probability of inserting new requests up to about 12% with respect to ALNS. As for the results shown in Table 4, MLNS performs as well or slightly worse than LNS, RAND, and ALNS on small subproblems with up to 5 requests and large subproblems with up to 60 requests.

Figure 8(a) shows the percentage of time we drew between 1 to 9 algorithms, following the RF classifiers predictions. As it can be noted, for subproblems of size 15 or less, we mostly only selected one algorithm. For larger subproblems, we mostly drew between two algorithms and up to four algorithms. Figure 8(b) shows the percentage of time algorithm 1 to 9 was drawn from the pool of predicted algorithms. As can be seen, for subproblems of size 15 or less, we mostly selected algorithm 3, i.e., random destroy and regret-3 repair heuristics. For larger subproblems, we mostly selected between algorithm 2 and algorithm 3, i.e. random destroy, regret-3 or regret-2 repair heuristics. However, for subproblems up to size 40 we also select the remaining algorithms with a probability of up to 11 percent. The only exception is provided by algorithm 9, i.e. Shaw removal and regret-3 repair heuristics, which is never selected.

The MLNS was designed for static e-ADARP subproblems that need to be efficiently solved in the context of dynamic operations. As such, we repeat tests by imposing an exit rule after 5 seconds have elapsed from the beginning of the search. Table 6 shows the percentage of times LNS, RAND, ALNS, and MLNS produced the highest-quality solution, based on the objective function value and within a limit of 5 seconds. As it can be noted, in this case, MLNS is selected a higher number of times for subproblems with more than 35 requests,

Table 6: Percentage of times LNS, RAND, ALNS, MLNS produces the highest-quality solution, based on the subproblem size and within a limit of 5 seconds

Size	#Subproblems	NONE [%]	LNS [%]	RAND [%]	ALNS [%]	MLNS [%]
5	18	11.11	27.78	38.89	22.22	0.00
10	933	30.44	23.90	8.15	22.62	14.90
15	3148	33.42	22.65	1.97	25.54	16.42
20	7075	31.69	23.63	1.53	25.48	17.67
25	8597	38.25	20.74	1.61	22.18	17.23
30	9882	32.98	22.19	1.56	23.27	20.00
35	12553	32.13	20.66	2.17	21.56	23.48
40	18298	43.08	15.95	2.47	16.70	21.81
45	18915	52.42	12.80	2.77	12.86	19.15
50	9071	60.09	10.40	2.73	9.21	17.57
55	2209	65.91	8.96	2.17	8.15	14.80
60	118	71.19	5.93	3.39	6.78	12.71

Table 7: Expected difference in the objective function improvement based on the subproblem size and and within a limit of 5 seconds: MLNS - LNS, MLNS - RAND, MLNS - ALNS

Size	#Subproblems	MLNS-LNS [%]	MLNS-RAND [%]	MLNS-ALNS [%]
5	16	4.87	-0.01	-0.01
10	649	2.32	-0.85	-0.77
15	2096	21.51	-2.19	-1.90
20	4833	24.23	-1.48	-2.27
25	5309	16.39	-1.87	-2.64
30	6623	20.03	-2.94	-2.81
35	8520	18.56	-1.56	-1.67
40	10416	14.86	0.12	-0.25
45	9000	10.19	0.80	0.96
50	3620	6.29	1.76	1.93
55	753	3.14	0.34	-0.15
60	34	0.19	0.13	0.19

a comparable amount of times for subproblems with about 30 requests, and lower amount of times for subproblems with up to 25 requests, respect to the competing search mechanisms. This is also confirmed by the results contained in Table 7, which show that MLNS produces results of slightly higher or comparable quality for subproblems with more than 40 requests, and of slightly lower quality (up to about -3 percent) for smaller subproblems, when enforcing a time limit of 5 seconds. However, note that MLNS is computationally more expensive than the competing search mechanisms, because of the prediction problem being solved at every iteration during the search.

As shown in Table 8, the MLNS search may be reduced by up to about

Table 8: Expected difference in the number of iterations based on the subproblem size and and within a limit of 5 seconds: MLNS - LNS, MLNS - RAND, MLNS - ALNS

Size	#Subproblems	MLNS-LNS	MLNS-RAND	MLNS-ALNS
5	16	-61.62	-61.62	-61.62
10	649	-56.19	-56.26	-58.91
15	2096	-58.11	-57.68	-59.43
20	4833	-45.82	-45.18	-46.38
25	5309	-36.04	-32.38	-33.30
30	6623	-27.49	-22.39	-23.28
35	8520	-18.71	-14.84	-15.11
40	10416	-12.75	-9.04	-9.29
45	9000	-8.37	-5.13	-5.27
50	3620	-5.73	-2.71	-2.79
55	753	-4.47	-1.59	-1.54
60	34	-2.50	-0.41	-0.38

Table 9: Expected difference in the number of incumbent solutions based on the subproblem size and and within a limit of 5 seconds: MLNS - LNS, MLNS - RAND, MLNS - ALNS

Size	#Subproblems	MLNS-LNS	MLNS-RAND	MLNS-ALNS
5	16	0.12	0.00	0.00
10	649	0.83	-0.21	-0.13
15	2096	1.84	-0.39	-0.40
20	4833	1.87	-0.55	-0.61
25	5309	1.80	-0.41	-0.57
30	6623	1.73	-0.37	-0.45
35	8520	1.36	-0.03	-0.10
40	10416	1.04	0.20	0.15
45	9000	0.77	0.24	0.22
50	3620	0.65	0.29	0.30
55	753	0.53	0.23	0.24
60	34	0.53	0.35	0.26

Table 10: Percentage of times LNS, RAND, ALNS, MLNS insert new requests, based on the subproblem size and within a time limit of 5 seconds

Problem size	#Sub-problems	LNS [%]	RAND [%]	ALNS [%]	MLLNS [%]
5	16	93.75	100.0	100.0	100.0
10	649	3.08	7.40	7.24	6.32
15	2096	15.46	47.90	47.52	44.94
20	4833	26.30	60.75	61.87	58.76
25	5309	34.87	59.58	60.63	57.04
30	6623	30.77	61.09	60.89	57.15
35	8520	26.58	53.40	53.52	51.22
40	10416	19.87	40.32	40.85	40.27
45	9000	16.43	29.74	29.47	30.70
50	3620	14.34	20.91	20.66	23.29
55	753	10.76	14.87	15.67	15.27
60	34	5.88	5.88	5.88	5.88

60 iterations on small-scale subproblems while maintaining a similar number of iterations on large-scale subproblems. This reduction is not irrelevant in the context of searches bound by a 5 seconds time limit. Furthermore, we note that the reduction in the number of iterations monotonously decrease with the subproblem size. Indeed, the computational complexity added by the machine learning predictors is mostly interesting for larger subproblems. For these problems, and provided the quality of MLNS results, the machine learning module is capable of positively directing the search, which features a relatively higher number of incumbent solutions as shown in Table 9. This is also confirmed by the results contained in Table 10, which show that MLNS increases the percentage of inserted requests for subproblems with more than 40 requests, and slighly decreases the percentage of inserted requests (up to about -3 percent) for smaller subproblems, when enforcing a time limit of 5 seconds.

In our implementation, the computational time for the prediction problem solved by MLNS is about 0.13 seconds per iteration, with only 0.01 seconds used for estimating the subproblem features. As such, MLNS is constrained by the latency time of the python library ScikitLearn[5]. The latency time can be significantly reduced by considering other ML approaches whose libraries are implemented in faster programming languages (e.g., XGBoost).

[5]For a discussion on computational performance of the ScikitLearn library see `https://scikit-learn.org/0.15/modules/computational_performance.html`

9. Summary

This work proposes a new data-driven extension to the family of large neighborhood search metaheuristics for autonomous ridesharing operations. The proposed metaheuristic employs a machine learning approach to direct the search of consecutive static subproblems which are encountered in the course of the autonomous ridesharing operations. Namely, the machine learning module predicts whether each of the nine destroy-repair couples are to be employed at the current iteration during the search. The training dataset is produced through simulation on synthetic instances extracted from real ridesharing data and is composed of more than one and a half million examples of LNS moves. Random forest classification models are trained and tested for each of the nine destroy-repair couples. Results show that the learned models are capable of reproducing the observed data with high confidence. Furthermore, the employed classification framework allows to extract most important problem features which correlate with the choice of destroy-repair operators during the search.

The models have been re-employed on a validation dataset composed of about 90,000 static subproblems. Results show that MLNS outperforms state-of-the-art metaheuristics by up to about 28 percent, on average, on subproblems with up to 60 requests. The capabilities of MLNS are favorable for larger-scale subproblems when enforcing a very strict time limit for the optimization process. Although MLNS features a reduced number of iterations given the computational burden of the prediction problem being solved at each iteration, MLNS is capable to direct the search towards positive areas thus encountering a higher number of incumbent solutions during the optimization process.

Finally, there are three main drawbacks in the adoption of a machine learning-based mechanism rather than an adaptive mechanism. First, historical data from LNS iterations on past problems may not be readily available and needs to be specifically produced for the given routing problem, through computationally-expensive simulations. Second, instances from the given routing problem need to be characterized by features that can only be selected through expert knowledge. Third, the adopted ML predictor needs to be appropriately calibrated through computationally expensive experiments during the offline training phase. Future work may include: (1) the adoption of a look-ahead policy to partially guide the optimization algorithm by future demand arrivals, (2) an analysis on the impact of the re-optimization frequency, (3) a comparison between service levels when adopting large and local neighborhood moves, and (4) the adoption of other classification frameworks from the machine learning literature.

Acknowledgments

The authors would like to thank Dan Assouline, Tim Hillel, Gael Lederrey, and Nicholas Molyneaux for the several brainstorming talks about machine learning and its applications in the context of our proposed work. The authors would also like to thank George Adaimi for helping with the setup for the HPC facilities at EPFL.

References

Albareda-Sambola, M., Fernández, E., & Laporte, G. (2014). The dynamic multiperiod vehicle routing problem with probabilistic information. *Computers & Operations Research*, *48*, 31–39.

Alonso-Mora, J., Samaranayake, S., Wallar, A., Frazzoli, E., & Rus, D. (2017). On-demand high-capacity ride-sharing via dynamic trip-vehicle assignment. *Proceedings of the National Academy of Sciences*, *114*, 462–467.

Archetti, C., Fernández, E., & Huerta-Muñoz, D. L. (2018). A two-phase solution algorithm for the flexible periodic vehicle routing problem. *Computers & Operations Research*, *99*, 27–37.

Attanasio, A., Cordeau, J.-F., Ghiani, G., & Laporte, G. (2004). Parallel tabu search heuristics for the dynamic multi-vehicle dial-a-ride problem. *Parallel Computing*, *30(3)*, 377–387.

Bengio, Y., Lodi, A., & Prouvost, A. (2021). Machine learning for combinatorial optimization: A methodological tour d'horizon. *European Journal of Operational Research*, *290*, 405–421.

Beojone, C. V., & Geroliminis, N. (2021). On the inefficiency of ride-sourcing services towards urban congestion. *Transportation Research Part C: emerging technologies*, *124*, 102890.

Berbeglia, G., Cordeau, J.-F., & Laporte, G. (2010). Dynamic pickup and delivery problems. *European Journal of Operational Research*, *202*, 8–15.

Bergstra, J., & Bengio, Y. (2012). Random search for hyper-parameter optimization. *Journal of Machine Learning Research*, *13(2)*, 281–305.

Bertsimas, D., & Dunn, J. (2017). Optimal classification trees. *Machine Learning*, *106*, 1039–1082.

Bertsimas, D., Jaillet, P., & Martin, S. (2019). Online vehicle routing: The edge of optimization in large-scale applications. *Operations Research*, *67*, 143–162.

Bertsimas, D., & Shioda, R. (2007). Classification and regression via integer optimization. *Operations Research*, *55*, 252–271.

Bonami, P., Lodi, A., & Zarpellon, G. (2018). Learning a classification of mixed-integer quadratic programming problems. In *International Conference on the Integration of Constraint Programming, Artificial Intelligence, and Operations Research* (pp. 595–604). Springer.

Bonfietti, A., Lombardi, M., & Milano, M. (2015). Embedding decision trees and random forests in constraint programming. In *International Conference on Integration of Constraint Programming, Artificial Intelligence, and Operations Research* (pp. 74–90). Springer.

Bongiovanni, C. (2020). *The Electric Autonomous Dial-a-Ride Problem*. Ph.D. thesis École Polytechnique Fédérale de Lausanne (EPFL). Available at: `https://infoscience.epfl.ch/record/279988?ln=en`.

Bongiovanni, C., Kaspi, M., & Geroliminis, N. (2019). The electric autonomous dial-a-ride problem. *Transportation Research Part B: Methodological*, *122*, 436–456.

Braekers, K., Caris, A., & Janssens, G. K. (2014). Exact and meta-heuristics approach for a general heterogeneous dial-a-ride problem with multi depots. *Transportation Research Part B: Methodological*, *67*, 166–186.

Breiman, L. (1996). Bagging predictors. *Machine Learning*, *24*, 123–140.

Breiman, L. (2001). Random forests. *Machine Learning*, *45*, 5–32.

Breiman, L. (2017). *Classification And Regression Trees*. Routledge.

Burke, E. K., Gendreau, M., Hyde, M., Kendall, G., Ochoa, G., Özcan, E., & Qu, R. (2013). Hyper-heuristics: A survey of the state of the art. *Journal of the Operational Research Society*, *64*, 1695–1724.

Chen, C., Liaw, A., & Breiman, L. (2004). Using random forest to learn imbalanced data. Technical Report, University of California Berkeley.

Chen, X., & Tian, Y. (2019). Learning to perform local rewriting for combinatorial optimization. *arXiv preprint arXiv:1810.00337v5.*

Cordeau, J. F., & Laporte, G. (2003). A tabu search heuristic for the static multi-vehicle dial-a-ride problem. *Transportation Research Part B: Methodological*, *37(6)*, 579–594.

Cordeau, J. F., & Laporte, G. (2007). The dial-a-ride probelm: models and algorithms. *Annals of Operations Research*, *153*, 29–46.

Cordeau, J.-F., Laporte, G., & Mercier, A. (2001). A unified tabu search heuristic for vehicle routing problems with time windows. *Journal of the Operational Research Society*, *52*, 928–936.

Coslovich, L., Pesenti, R., & Ukovich, W. (2006). A two-phase insertion technique of unexpected customers for a dynamic dial-a-ride problem. *European Journal of Operational Research*, *175(3)*, 1605–1615.

Diana, M., & Dessouky, M. M. (2004). A new regret insertion heuristic for solving large-scale dial-a-ride problems with time windows. *Transportation Research Part B*, *38*, 539–557.

Dunning, I., Huchette, J., & Lubin, M. (2017). Jump: A modeling language for mathematical optimization. *SIAM Review*, *59*, 295–320.

Elshaer, R., & Awad, H. (2020). A taxonomic review of metaheuristic algorithms for solving the vehicle routing problem and its variants. *Computers & Industrial Engineering*, *140*, 106242.

Ferrucci, F., & Bock, S. (2015). A general approach for controlling vehicle en-route diversions in dynamic vehicle routing problems. *Transportation Research Part B*, *77*, 76–87.

Ferrucci, F., & Bock, S. (2016). Pro-active real-time routing in applications with multiple request patterns. *European Journal of Operational Research*, *253*, 356–371.

Ferrucci, F., Bock, S., & Gendreau, M. (2013). A pro-active real-time control approach for dynamic vehicle routing problems dealing with the delivery of urgent goods. *European Journal of Operational Research*, *225*, 130–141.

Funke, B., Grünert, T., & Irnich, S. (2005). Local search for vehicle routing and scheduling problems: Review and conceptual integration. *Journal of Heuristics*, *11*, 267–306.

Gao, L., Chen, M., Chen, Q., Luo, G., Zhu, N., & Liu, Z. (2020). Learn to design the heuristics for vehicle routing problem. *arXiv preprint arXiv:2002.08539*.

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

Gomes, C. P., & Selman, B. (2001). Algorithm portfolios. *Artificial Intelligence*, *126*, 43–62.

Gschwind, T., & Drexl, M. (2019). Adaptive large neighborhood search with a constant-time feasibility test for the dial-a-ride problem. *Transportation Science*, *53*, 480–491.

Gunluk, O., Kalagnanam, J., Menickelly, M., & Scheinberg, K. (2016). Optimal generalized decision trees via integer programming. *arXiv preprint arXiv:1612.03225*.

Hastie, T., Tibshirani, R., & Friedman, J. (2009). *The Elements of Statistical Learning* volume 2. Springer.

Ho, S. C., Szeto, W. Y., Kuo, Y.-H., Leung, J. M., Petering, M., & Tou, T. W. (2018). A survey of dial-a-ride problems: Literature review and recent developments. *Transportation Research Part B: Methodological*, *111*, 395–421.

Hottung, A., & Tierney, K. (2019). Neural large neighborhood search for the capacitated vehicle routing problem. *arXiv preprint arXiv:1911.09539*.

Hyland, M., & Mahmassani, H. S. (2020). Operational benefits and challenges of shared-ride automated mobility-on-demand services. *Transportation Research Part A: Policy and Practice*, *134*, 251–270.

Ichoua, S., Gendreau, M., & Potvin, J.-Y. (2006). Exploiting knowledge about future demands for real-time vehicle dispatching. *Transportation Science*, *40(2)*, 211–225.

Jaw, J.-J., Odoni, A. R., Psaraftis, H. N., & Wilson, N. H. (1986). A heuristic algorithm for the multi-vehicle advance request dial-a-ride problem with time windows. *Transportation Research Part B: Methodological*, *20*, 243–257.

Jun, S., Lee, S., & Chun, H. (2019). Learning dispatching rules using random forest in flexible job shop scheduling problems. *International Journal of Production Research*, *57*, 3290–3310.

Kerschke, P., Hoos, H. H., Neumann, F., & Trautmann, H. (2019). Automated algorithm selection: Survey and perspectives. *Evolutionary Computation*, *27*, 3–45.

Kruber, M., Lübbecke, M. E., & Parmentier, A. (2017). Learning when to use a decomposition. In *International Conference on AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems* (pp. 202–210). Springer.

Larsen, E., Lachapelle, S., Bengio, Y., Frejinger, E., Lacoste-Julien, S., & Lodi, A. (2021). Predicting tactical solutions to operational planning problems under imperfect information. *INFORMS Journal on Computing*, *0(0)*.

Lerman, R. I., & Yitzhaki, S. (1984). A note on the calculation and interpretation of the gini index. *Economics Letters*, *15*, 363–368.

Li, B., Krushinsky, D., Van Woensel, T., & Reijers, H. A. (2016). An adaptive large neighborhood search heuristic for the share-a-ride problem. *Computers & Operations Research*, *66*, 170–180.

Liaw, A., Wiener, M. et al. (2002). Classification and regression by randomforest. *R News*, *2*, 18–22.

Liu, T., Krishnakumari, P., & Cats, O. (2019). Exploring demand patterns of a ride-sourcing service using spatial and temporal clustering. In *2019 6th International Conference on Models and Technologies for Intelligent Transportation Systems (MT-ITS)* (pp. 1–9). IEEE.

Lodi, A., & Zarpellon, G. (2017). On learning and branching: a survey. *Top*, *25*, 207–236.

Lu, H., Zhang, X., & Yang, S. (2019). A learning-based iterative method for solving vehicle routing problems. In *International Conference on Learning Representations (ICLR)*.

Malitsky, Y., Sabharwal, A., Samulowitz, H., & Sellmann, M. (2013). Algorithm portfolios based on cost-sensitive hierarchical clustering. In *Twenty-Third International Joint Conference on Artificial Intelligence*.

Marković, N., Nair, R., Schonfeld, P., Miller-Hooks, E., & Mohebbi, M. (2015). Optimizing dial-a-ride services in maryland: benefits of computerized routing and scheduling. *Transportation Research Part C: Emerging Technologies*, *55*, 156–165.

Masmoudi, M. A., Hosny, M., Braekers, K., & Dammak, A. (2016). Three effective metaheuristics to solve the multi-depot multi-trip heterogeneous dial-a-ride problem. *Transportation Research Part E: Logistics and Transportation Review*, *96*, 60–80.

Molenbruch, Y., Braekers, K., Caris, A., & Berghe, G. V. (2017). Multi-directional local search for a bi-objective dial-a-ride problem in patient transportation. *Computers & Operations Research*, *77*, 58–71.

Nabian, M. A., Alemazkoor, N., & Meidani, H. (2019). Predicting near-term train schedule performance and delay using bi-level random forests. *Transportation Research Record*, *2673*, 564–573.

Nikolaev, A. G., & Jacobson, S. H. (2010). Simulated annealing. In *Handbook of Metaheuristics* (pp. 1–39). Springer.

Parragh, S. N. (2011). Introducing heterogeneous users and vehicles into models and algorithms for the dial-a-ride problem. *Transportation Research Part C: Emerging Technologies*, *19(5)*, 912–930.

Parragh, S. N., Doerner, K. F., & Hartl, R. F. (2010). Variable neighborhood search for the dial-a-ride problem. *Computers & Operations Research*, *37*, 1129–1138.

Parragh, S. N., Doerner, K. F., Hartl, R. F., & Gandibleux, X. (2009). A heuristic two-phase solution approach for the multi-objective dial-a-ride problem. *Networks: An International Journal*, *54*, 227–242.

Peled, I., Lee, K., Jiang, Y., Dauwels, J., & Pereira, F. C. (2019). Online predictive optimization framework for stochastic demand-responsive transit services. *arXiv preprint arXiv:1902.09745*.

Pelletier, S., Jabali, O., Laporte, G., & Veneroni, M. (2017). Battery degradation and behaviour for electric vehicles: Review and numerical analyses of several models. *Transportation Research Part B: Methodological*, *103*, 158–187.

Pisinger, D., & Ropke, S. (2010). Large neighborhood search. In *Handbook of Metaheuristics* (pp. 399–419). Springer.

Potvin, J.-Y., & Rousseau, J.-M. (1993). A parallel route building algorithm for the vehicle routing and scheduling problem with time windows. *European Journal of Operational Research*, *66*, 331–340.

Ropke, S., & Pisinger, D. (2006). An adaptive large neighborhood search heuristic for the pickup and delivery problem with time windows. *Transportation Science*, *40*, 455–472.

Sacramento, D., Pisinger, D., & Ropke, S. (2019). An adaptive large neighborhood search metaheuristic for the vehicle routing problem with drones. *Transportation Research Part C: Emerging Technologies*, *102*, 289–315.

Savelsbergh, M. W. (1992). The vehicle routing problem with time windows: Minimizing route duration. *ORSA Journal on Computing*, *4*, 146–154.

Schilde, M., Doerner, K., & Hartl, R. (2011a). Metaheuristics for the dynamic stochastic dial-a-ride problem with expected return transports. *Computers & Operations Research*, *38(12)*, 1719–1730.

Schilde, M., Doerner, K. F., & Hartl, R. F. (2011b). Metaheuristics for the dynamic stochastic dial-a-ride problem with expected return transports. *Computers & Operations Research*, *38*, 1719–1730.

Schilde, M., Doerner, K. F., & Hartl, R. F. (2014). Integrating stochastic time-dependent travel speed in solution methods for the dynamic dial-a-ride problem. *European Journal of Operational Research*, *238*, 18–30.

Shaw, P. (1997). A new local search algorithm providing high quality solutions to vehicle routing problems. *APES Group, Dept of Computer Science, University of Strathclyde, Glasgow, Scotland, UK*, *46*.

Simonetto, A., Monteil, J., & Gambella, C. (2019). Real-time city-scale ridesharing via linear assignment problems. *Transportation Research Part C: Emerging Technologies*, *101*, 208–232.

Vinyals, O., Fortunato, M., & Jaitly, N. (2015). Pointer networks. In *Advances in Neural Information Processing Systems* (pp. 2692–2700).

Weisstein, E. W. (2020). Law of cosines. Available at: `https://mathworld.wolfram.com/LawofCosines.html`.

Wu, D., Jennings, C., Terpenny, J., Gao, R. X., & Kumara, S. (2017). A comparative study on machine learning algorithms for smart manufacturing: tool wear prediction using random forests. *Journal of Manufacturing Science and Engineering*, *139(7)*, 1–9.

Wu, Y., Song, W., Cao, Z., Zhang, J., & Lim, A. (2021). Learning improvement heuristics for solving routing problems. *IEEE Transactions on Neural Networks and Learning Systems*, (pp. 1–13).

Zalesak, M., & Samaranayake, S. (2021). Real time operation of high-capacity electric vehicle ridesharing fleets. *arXiv preprint arXiv:2101.01274*.

Zhang, X., Chen, L., Gendreau, M., & Langevin, A. (2021). Learning-based branch-and-price algorithms for the vehicle routing problem with time windows and two-dimensional loading constraints. *INFORMS Journal on Computing*, *0(0)*.

Appendix A. MILP for static e-ADARP subproblems

At time $h \leq H$, the transportation system receives a new request with pickup location $\tilde{p}$ and dropoff location $\tilde{d}$. No information regarding the request location is available beforehand. However, its booking time may be issued a few minutes in advance with respect to the desired pickup time. If more than one new transport request appears at the same time, the requests are processed independently and in the order of arrival. Depending on the current state and future tasks assigned to the vehicles, the new request $\{\tilde{p}, \tilde{d}\}$ may be either accepted or rejected. Service is only rejected if the new request cannot be feasibly inserted into the existing vehicle routes, in consideration of the constraints imposed by the e-ADARP problem. Denying service to the new request incurs a fixed cost penalty c_3, which affects the objective function through an auxiliary binary decision variable $z_{\tilde{p}}$. This cost penalty c_3 may be computed as a proportion of the total routing cost or as a high fixed cost. As such, existing vehicle plans may need to be revisited in order to find the optimal trade-off between the following components of the objective function: (1) feasibly insert the new request, (2) decrease operational cost, (3) and increase the level of service. This is obtained by re-optimizing a current static subproblem at time h, which is composed of all information related to the current vehicle locations $\bar{o}^k$, the current vehicle plans, and the new request.

Upon the arrival of a new request, each vehicle $k \in \mathcal{K}$ may be traveling or have just arrived at a location $\bar{j}^k$ corresponding to a pickup in $\mathcal{P}$, a dropoff in $\mathcal{D}$, a charging station in $\mathcal{S}$, or a destination depot in $\mathcal{F}$. As such, vehicle travel times from the current vehicle locations $\bar{o}^k$ to $\bar{j}^k$, i.e., $t_{\bar{o}^k,\bar{j}^k}$, are computed to account for the remaining portion of the route to be traveled. Direct deviation times $t_{\bar{o}^k,\tilde{p}}$ from the vehicle current locations to the pickup of the new request are also simply computed by employing the law of cosines (Weisstein, 2020). Services that already started at pickup or dropoff locations cannot be preempted. However, vehicles that are waiting at pickup or dropoff nodes can be rerouted to other locations. Note that parking spaces may not be available at locations $j \in \mathcal{N}$. Thus, vehicle waiting times are bounded from above (e.g., by a few minutes) through a maximum waiting time w_j. In contrast, charging stages at stations $s \in \mathcal{S}$ can be anticipated if the vehicle charge level is sufficient to allow travel between locations.

Upon the arrival of a new request $\{\tilde{p}, \tilde{d}\}$, vehicles may be non-empty. In particular, some requests with pickups $\hat{\mathcal{P}}^k \subseteq P$ and dropoffs $\hat{\mathcal{D}}^k \subseteq \mathcal{D}$ may have been picked-up by vehicle $k \in \hat{\mathcal{K}} \subseteq \mathcal{K}$ without being dropped-off. In this case, the initial occupancy of all of such vehicles is to be reduced by the number of requests currently on board. From a modeling perspective, dummy pickup nodes $\hat{\mathcal{P}}^k$ are generated at the corresponding vehicle locations and need to be served. In order to produce routing solutions starting with sequential visits all of the dummy pickups $\hat{\mathcal{P}}^k$, the following inequalities are enforced:

$$\sum_{i=1}^{|\hat{\mathcal{P}}^k|-1} x^k_{o^k,i} + x^k_{i,i+1} = |\hat{\mathcal{P}}^k| - 1 \qquad \forall k \in \hat{\mathcal{K}}.$$

Note that the order in which dummy pickup nodes $\hat{\mathcal{P}}^k$ are served is not relevant. Furthermore, the maximum ride time and the direct travel time $t_{i,n+i}$ of each request in $i \in \hat{\mathcal{P}}^k$ are both reduced by the time already traveled on board of vehicle k, as follows:

$$u_i = u_i - (h - T_i^k) \qquad \forall k \in \hat{\mathcal{K}}, i \in \hat{\mathcal{P}}^k$$

$$t_{i,n+i} = t_{i,n+i} - (h - T_i^k) \qquad \forall k \in \hat{\mathcal{K}}, \forall i \in \hat{\mathcal{P}}^k.$$

With such pre-processing steps, static e-ADARP subproblems at time h can be formulated as the following 3-indexed MILP, based on Bongiovanni et al. (2019):

$$\min c_1 \cdot \sum_{k\in\mathcal{K}} \sum_{i,j\in\mathcal{V}} t_{ij} x_{ij}^k + c_2 \cdot \sum_{i\in\mathcal{P}} R_i + c_3 \cdot \sum_{i\in\mathcal{P}\cup\tilde{p}} z_i \tag{A.1}$$

subject to:

$$\sum_{j\in\mathcal{P}\cup\mathcal{S}\cup\mathcal{F}\cup\{\tilde{p}\}} x_{o^k j}^k = 1 \qquad \forall k \in \mathcal{K} \tag{A.2}$$

$$\sum_{j\in\mathcal{F}} \sum_{i\in\mathcal{D}\cup\mathcal{S}\cup\{\tilde{d}\}} x_{ij}^k = 1 \qquad \forall k \in \mathcal{K} \tag{A.3}$$

$$\sum_{k\in\mathcal{K}} \sum_{\substack{j\in\mathcal{N}\cup\{\tilde{p},\tilde{d}\} \\ j\neq i}} x_{ij}^k = 1 \qquad \forall i \in \mathcal{P} \tag{A.4}$$

$$\sum_{k\in\mathcal{K}} \sum_{i\in\mathcal{V}\setminus\mathcal{F}} x_{i,\tilde{p}}^k \leq 1 \tag{A.5}$$

$$\sum_{k\in\mathcal{K}} \sum_{i\in\mathcal{V}\setminus\mathcal{F}} x_{ij}^k = 1 - z_j \qquad \forall j \in \mathcal{P} \cup \tilde{p} \tag{A.6}$$

$$\sum_{k\in\mathcal{K}} \sum_{i\in\{o^k\}\cup\mathcal{D}\cup\{\tilde{d}\}\cup\mathcal{S}} x_{ij}^k \leq 1 \qquad \forall j \in \mathcal{F} \cup \mathcal{S} \tag{A.7}$$

$$\sum_{\substack{j\in\mathcal{V}\cup\{\tilde{p},\tilde{d}\} \\ j\neq i}} x_{ij}^k - \sum_{\substack{j\in\mathcal{V}\cup\{\tilde{p},\tilde{d}\} \\ j\neq i}} x_{j,i}^k = 0 \qquad \forall k \in \mathcal{K}, i \in \mathcal{N} \cup \mathcal{S} \cup \{\tilde{p}, \tilde{d}\} \tag{A.8}$$

$$\sum_{\substack{j\in\mathcal{N}\cup\{\tilde{p},\tilde{d}\} \\ j\neq i}} x_{ij}^k - \sum_{\substack{j\in\mathcal{N}\cup\{\tilde{p},\tilde{d}\} \\ j\neq n+i}} x_{j,n+i}^k = 0 \qquad \forall k \in \mathcal{K}, i \in \mathcal{P} \cup \{\tilde{p}\} \tag{A.9}$$

$$T_i^k + d_i + t_{i,n+i} \leq T_{n+i}^k \qquad \forall k \in \mathcal{K}, i \in \mathcal{P} \cup \{\tilde{p}\} \tag{A.10}$$

$$arr_i \leq T_i^k \leq dep_i \qquad \forall k \in \mathcal{K}, i \in \mathcal{V} \cup \{\tilde{p}, \tilde{d}\} \tag{A.11}$$

$$T_{n+i}^k - T_i^k - d_i \leq u_i \qquad \forall k \in \mathcal{K}, i \in \mathcal{P} \cup \{\tilde{p}, \tilde{d}\} \tag{A.12}$$

$$T_i^k + t_{ij} + d_i - M_{i,j}^k (1 - x_{ij}^k) \leq T_j^k \qquad \forall k \in \mathcal{K}, i \in \mathcal{V} \cup \{\tilde{p}, \tilde{d}\}, j \in \mathcal{V} \cup \{\tilde{p}, \tilde{d}\}, i \neq j | M_{i,j}^k > 0 \tag{A.13}$$

$$x_{ij}^k = 1 \Rightarrow T_j^k - (T_i^k + t_{ij} + d_i) \leq w_j \qquad \forall k \in \mathcal{K}, i \in \mathcal{N} \cup \{\tilde{p}, \tilde{d}\}, j \in \mathcal{N} \cup \{\tilde{p}, \tilde{d}\}, i \neq j \tag{A.14}$$

$$R_i \geq T_{n+i}^k - T_i^k - d_i - t_{i,n+i} \qquad \forall k \in \mathcal{K}, i \in \mathcal{P} \cup \{\tilde{p}\} \tag{A.15}$$

$$L_i^k = 0 \qquad \forall k \in \mathcal{K}, i \in o^k \tag{A.16}$$

$$L_i^k \geq \max(0, l_i) \qquad \forall k \in \mathcal{K}, \forall i \in \mathcal{N} \cup \{\tilde{p}, \tilde{d}\} \tag{A.17}$$

$$L_i^k \leq \min(C^k, C^k + l_i) \qquad \forall k \in \mathcal{K}, \forall i \in \mathcal{N} \cup \{\tilde{p}, \tilde{d}\} \tag{A.18}$$

$$L_i^k + l_j - G_{i,j}^k (1 - x_{ij}^k) = L_j^k \qquad \forall k \in \mathcal{K}, i \in \mathcal{V} \cup \{\tilde{p}, \tilde{d}\}, j \in \mathcal{V} \cup \{\tilde{p}, \tilde{d}\}, i \neq j | G_{i,j}^k > 0 \tag{A.19}$$

$$L_i^k = 0 \qquad \forall k \in \mathcal{K}, i \in \mathcal{F} \cup \mathcal{S} \tag{A.20}$$

$$B_i^k = B_0^k \qquad \forall k \in \mathcal{K}, i \in o^k \tag{A.21}$$

$$B_j^k = B_i^k - \beta_{i,j} + Q(1 - x_{ij}^k) \qquad \forall k \in \mathcal{K}, i \in \mathcal{V} \cup \{\tilde{p}, \tilde{d}\} \setminus \mathcal{S}, j \in \mathcal{V} \cup \{\tilde{p}, \tilde{d}\} \setminus \{o^k\}, i \neq j | Q > 0 \tag{A.22}$$

$$B_j^k = B_s^k + \alpha_s E_s^k - \beta_{s,j} + Q(1 - x_{s,j}^k) \qquad \forall k \in \mathcal{K}, s \in \mathcal{S}, j \in \mathcal{P} \cup \{\tilde{p}\} \cup \mathcal{F} \cup \mathcal{S}, s \neq j \tag{A.23}$$

$$E_s^k = T_s^k - t_{i,s} - T_i^k + M_{i,s}^k(1 - x_{i,s}^k) \qquad \forall k \in \mathcal{K}, \forall s \in \mathcal{S}, i \in \mathcal{D} \cup \{\tilde{d}\} \cup \mathcal{S} \cup \{o^k\}, i \neq s | Mi, s^k > 0 \tag{A.24}$$

$$Q \geq B_s^k + \alpha_s E_s^k \qquad \forall k \in \mathcal{K}, s \in \mathcal{S} \tag{A.25}$$

$$B_i^k \geq rQ \qquad \forall k \in \mathcal{K}, i \in \mathcal{F} \tag{A.26}$$

$$x_{ij}^k \in \{0, 1\} \qquad \forall k \in \mathcal{K}, i \in \mathcal{V} \cup \{\tilde{p}, \tilde{d}\}, j \in \mathcal{V} \cup \{\tilde{p}, \tilde{d}\} \tag{A.27}$$

$$B_i^k \geq 0 \qquad \forall k \in \mathcal{K}, i \in \mathcal{V} \cup \{\tilde{p}, \tilde{d}\} \tag{A.28}$$

$$E_s^k \geq 0 \qquad \forall k \in \mathcal{K}, s \in \mathcal{S}. \tag{A.29}$$

The objective function (A.1) minimizes the weighted-sum objective composed of three components, namely the total traveled distance, the total excess ride time, and the penalty for denying the new request. Note that the objective function pushes towards solutions including the new request if weight factor c_3 is sensibly larger than weight factors c_1 and c_2. Constraints (A.2)-(A.3) ensure that all vehicles exit the origin depots and return to one of the available destination depots. Constraints (A.4) ensure that all of the requests in the vehicle plan are served. Constraints (A.5) allow vehicles to serve the new request $\{\tilde{p}\}$. If the new request is served, the auxiliary decision variable $z_{\tilde{p}}$ is set to zero and the third component of the objective no longer contributes to the total cost (A.6). Note that feasible solutions only consider solutions with $z_i = 0$ for $i \in P$. However, during the solution process, relaxations of the problem may consider intermediate infeasible solutions with $z_i > 0 \ \forall i \in \mathcal{P}$. Charging stations may be accessed from dropoff locations or other charging stations (A.7). Constraints (A.8) ensure flow conservation, while constraints (A.9) ensure that the same vehicle serves both the pickup and the dropoff of each request. Furthermore, pickup-dropoff precedences are enforced in constraints (A.10). Time window and maximum ride time constraints are set in (A.11)-(A.12). Service start times between nodes are computed through constraints (A.13) and bounded at pickup/dropoff locations by their maximal wait times (A.14). User excess ride times are set in (A.15). Vehicle loads are initialized through constraints (A.16). Loads are bounded by constraints (A.17)-(A.18) and are updated between consequent nodes according to (A.19). Charging stations and destination depots may be only accessed by empty vehicles (A.20). Constraints (A.21) set the initial battery levels while the SOC between following nodes is tracked through constraints (A.22). Vehicle battery inventories can be re-increased after visiting a charging facility (A.23), where they may recharge according to (A.24) and up to the vehicle maximum battery capacity (A.25). Vehicles have to return to the selected destination depots with some minimal battery charge (A.26). Finally, (A.27)-(A.29) are integrality constraints. Note that constraints (A.13), (A.19), (A.24) are linearized through time-dependent and load-dependent big-M parameters which can be computed as proposed in Bongiovanni et al. (2019).

Appendix B. Event-based simulation framework

To represent realistic operations, the dynamic e-ADARP is simulated through an event-based simulation approach, depicted by the flowchart in Figure B.9. The input to the simulator comprises transportation network information (i.e., origin/destination depots, charging stations, travel times) and initial vehicle routes. Without loss of generality, we assume that there are no pre-booked requests and that the initial vehicle routes consist only of origin depots, charging stations, and destination depots.

Origin depots for vehicles are randomly selected from the set of available origin depots in the transportation network. Vehicles are not allowed to wait at origin depots and are instead allowed to idle at charging stations, before returning to one of the optional destination depots at the end of the planning horizon of duration H. Each vehicle chooses the charging station and destination depot that minimizes travel distance. A set of n dynamic requests appear during the planning horizon and are characterized by certain pickup and delivery locations, time windows, and stochastic reservation times. For each request $i = 1, \ldots, n$, the reservation time is obtained by generating an in advance booking time that is subtracted from the first arrival time arr_i. The booking times are computed as independent and identically distributed random variables from an exponential distribution with a rate parameter λ, which is assumed to be homogeneous across requests. Note that larger rate parameters λ induce instantaneous booking times.

The generated input enters the simulation framework, which considers vehicle events and request events. Specifically, the vehicle events consider departure, arrival, and charging events. The request events consider booking, pickup, and delivery events. The simulation is initialized by generating an event list consisting of vehicle departure events from the origin depots and an initial request arrival event. Vehicle departure events trigger vehicle arrival events at the following locations in the vehicle plans. Arrival events appear before any vehicle waiting phases, which are taken into account before proceeding to charging, pickup, or delivery events. Vehicle departure events are triggered after service (i.e., pickup, delivery, recharge). The arrival of each request is processed through a two-phase metaheuristic. If the request is accepted and inserted feasibly, the associated vehicle plan is updated. If the inserted request needs to be served immediately, the vehicle tasks can be modified, as well as the event list. There are two types of vehicle tasks that can be modified following the arrival of a new request. Vehicles that are currently waiting or recharging can depart instantly to provide service to the newly arrived request. Vehicles that are currently en route to a specific location can instantaneously divert to the newly arrived request. In the latter case, travel times are updated to account for the portion of the route that has already been traveled. Note that vehicles cannot instantaneously depart and provide service to new requests if they are providing service elsewhere. In addition, instantaneous departures are only allowed in accordance with time windows, maximum travel time, and vehicle capacity considerations for all requests. Each request arrival event triggers a next request arrival event, which is selected in the order of the stochastic reservation times generated as input.

The simulation ends after the return of each vehicle at the selected destination depots. Regarding the latter, note that the arrival of new requests may change the choice of destination depot for the vehicles providing service.

In this work, the stochasticity of the simulation is represented by the arrival of new requests. As such, the two-phase metaheuristic, and in particular the second re-optimization stage, is triggered only after the arrival of new requests. However, the proposed event-based simulation can be easily modified to trigger the second re-optimization phase in light of other changing conditions or optimization policies (e.g., traffic congestion, fixed-time polishing, etc.), or to simultaneously treat multiple request arrivals at once.

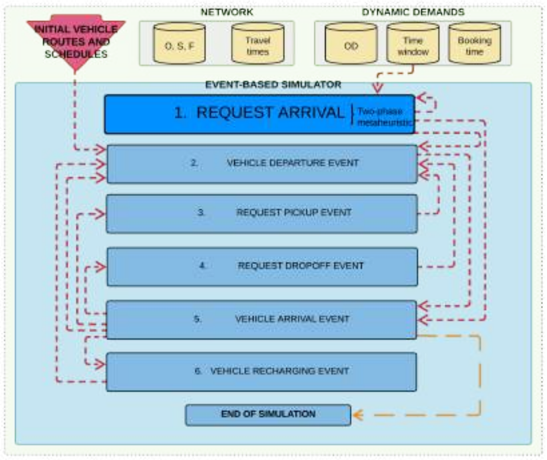


Figure B.9: Flowchart for the event-based simulation