Received 15 December 2022, accepted 30 December 2022, date of publication 3 January 2023, date of current version 6 January 2023.

Digital Object Identifier 10.1109/ACCESS.2022.3233852

RESEARCH ARTICLE

Learned Upper Bounds for the Time-Dependent Travelling Salesman Problem

TOMMASO ADAMO, GIANPAOLO GHIANI, PIERPAOLO GRECO, AND EMANUELA GUERRIERO

Department of Engineering for Innovation, University of Salento, 73100 Lecce, Italy

Corresponding author: Emanuela Guerriero (emanuela.guerriero@unisalento.it)

ABSTRACT Fleet management plays a central role in several application contexts such as distribution planning, mail delivery, garbage collection, salt gritting, field service routing. Since road congestion has a big impact on driving times, fleet management can be enhanced by taking into account data on current traffic conditions. Today, most carriers gather high-quality historical traffic data by using global position system information. These data serve as an input for defining time-dependent travel times, i.e. travel times changing according to traffic conditions throughout the day. Given a fixed-size fleet of vehicles and a graph with arc traversal times varying over time, Time-Dependent Vehicle Routing Problems aim to select the *best* routes while minimizing the travelling costs. The basic version with only one route is usually referred to as the Time-Dependent Travelling Salesman Problem. The main goal of this work is to define tight upper bounds for this problem by reusing the information gained when solving instances with similar features. This is customary in distribution management, where vehicle routes have to be generated over and over again with similar input data. To this aim, the authors devise an upper bounding technique based on the solution of a classical (and simpler) time-independent Asymmetric Travelling Salesman Problem, where the constant arc costs are suitably defined by the combined use of a Linear Program and a mix of unsupervised and supervised Machine Learning techniques. The effectiveness of this approach has been assessed through a computational campaign on the real travel time functions of two European cities: Paris and London. The overall average gap between the proposed heuristic and the best-known solutions is about 0.001%. For 31 instances, new best solutions have been obtained.

INDEX TERMS Machine learning, path ranking invariance, time-dependent routing, travelling salesman problem.

I. INTRODUCTION

The purpose of this article is to present a Machine Learning (ML) enhanced upper-bound for the *Time-Dependent Travelling Salesman Problem* (TDTSP), defined as follows. Let $G := (V \cup \{0\}, A, \tau)$ denote a time-dependent directed complete graph, where $V = \{1, \ldots, n\}$ is the set of customers, vertex 0 is the depot and $A := \{(i,j) : i \in V, j \in V\} \bigcup \{(0, i) : i \in V\} \bigcup \{(i, 0) : i \in V\}$ is the set of arcs. With each arc $(i,j) \in A$ is associated a travel time function $\tau_{ij}(t)$, representing the travel time of (i,j) if the vehicle leaves node i at time t. The TDTSP amounts to determine a least duration tour visiting each customer once, with the vehicle leaving the depot at time 0.

In recent years there has been a flourishing of scholarly works in time-dependent routing. In routing problems, travel time is a non linear function of average travel speed, which may vary exogenously or endogenously. Today, most carriers gather high-quality historical traffic data by using global position system information. These data serve as an input for defining travel times modelling travel speed changes due to exogenous events, like traffic congestion and weather conditions. On the other hand, in routing problems travel speeds may also vary endogenously whenever the decision maker can prescribe the vehicles' speeds, e.g. in order to take into account energy consumption [1] or CO_2 emissions [2]. The

The associate editor coordinating the review of this manuscript and approving it for publication was Zhenzhou Tang.

present contribution deals with time-dependent routing problems where time-varying travel time aims to model traffic conditions throughout the day. In the following, it has been presented a brief review of contributions related to TDTSP. For a complete survey see [3]. Reference [4] represented the first one to address the TDTSP and devised a *Mixed Integer Programming* (MIP) model. An approximate dynamic programming algorithm was proposed in [5], whereas two heuristics has been developed in [6]. A simulated annealing heuristic was proposed in [7] and some metaheuristics were proposed in [8]. In [9] the authors exploited some properties of the TDTSP, in order to develop a lower and upper bounding algorithm. Moreover the authors proposed a MIP model for which they devised valid inequalities. The separation procedures for the proposed inequalities were then embedded into a branch-and-cut algorithm that solved instances with up to 40 vertices. In [10], some properties of the problem are derived as well as a branch-and-bound algorithm. The computational campaign showed that the proposed approach outperforms the branch-and-cut procedure by [9]. Reference [11] proposed a *Constraint Programming* solution approach. This algorithm, thanks to new global constraints, was able to solve instances with up to 30 customers. Recently, a parameterized family of lower bounds has been proposed by [12], where the setting of parameters are carried out by fitting the traffic data. The performance of lower bounding mechanism was evaluated by embedding it in a branch-and-bound procedure. The computational campaign showed that it was possible to determine the optimal solution for a larger number of instances than [10]. Several contributions studied a variant of TDTSP with Time Windows (TDTSPTW). The approach proposed in [13] is based on a transformation of the TDTSPTW into an Asymmetric Generalized TSP and then into an Asymmetric Graphical TSP, solved by a known exact algorithm for the Mixed General Routing Problem. Contribution [14] aims to extend results provided in [9] to deal with time windows. The authors demonstrated that a lower bound and an upper bound for the original TDTSPTW can be derived from the optimal solution of an Asymmetric TSPTW with suitably defined travel times and time windows. The proposed bounds are integrated into an exact branch-and-bound algorithm. A new formulation and branch-and-cut algorithm is devised in [15]. Reference [16] proposed a solution approach relying on a dynamic discretization discovery framework, which is based on integer programming formulations defined on (partially) time expanded networks. Reference [17] deals with a heuristic solution algorithm for the TDTSPTW, named Iterated Maximum Large Neighborhood Search. The algorithm starts from a given solution, which tries to improve iteratively by applying destroy and repair operators. Some customers are then randomly shifted during a perturbation phase. Other contributions examine other variants of the TDTSP. In [18] exact and approximate algorithms are proposed for the *Moving-Target TSP*, where a set of targets, moving at constant speed, has to be intercepted in minimum time by a pursuer. Reference [19] addressed *the Robust TSP with Interval Data*, where travel times correspond to ranges of *possible* values. Finally, it is worth noting that there are contributions dealing with a scheduling problem referred to as TDTSP. Given a single machine and a set of jobs, it aims to determine a sequence of jobs, where the processing times are position-dependent. Such contributions are not relevant for the present contribution.

The contribution of this paper also lies at the boundary between machine learning and combinatorial optimization. Following the classification introduced in [20], there are different algorithmic structures, where learning components and OR algorithms can be laid out. It is worth noting that solving the TSP through ML is not new. Several contributions follows the *end-to-end learning* algorithmic structure, i.e. determine approximate TSP solution in a pure data-driven fashion by training the ML model to output solutions directly from the input instance. Reference [21] tackles Euclidean TSP with deep learning and introduces the pointer network wherein an encoder, namely a recurrent neural network, is used to parse nodes in the input graph and produces an encoding (a vector of activations) for each of them. Then a decoder predicts a policy for prescribing the next possible move so that to sample a permutation of visited cities. This method makes it possible to use the network over different input graph sizes. The authors train the model through supervised learning with precomputed TSP solutions as targets. A similar model is used in [22] and trained with reinforcement learning using the negative tour length as a reward signal. The authors discuss some limitations of supervised learning, such as the need to determine optimal TSP solutions (the targets), that in turn, may be ill-defined when those solutions are not optimal, or when there are multiple solutions. Reference [23] devised a three-step procedure, starting with a semantic feature extraction from the MIP model of the TSP. The extracted features are then exploited to derive a neighbourhood design mechanisms. Finally an automatic configuration phase finds the *proper mix* of such mechanisms taking into account the instance distribution. The contribution [24] provides a comparative analysis of ML-based heuristics for the classical (time-invariant) Travelling Salesman Problem. To the best of these authors' knowledge, contribution [25] is the only attempt to use ML to solve a time-dependent routing problem. In particular, the authors showed how to embed ML techniques in a simple constructive heuristic for the TDTSP. Computational results of [25] demonstrated that the proposed algorithmic approach is promising in real-time settings, where speed updates and/or arrivals of new requests may lead to re-optimization of the planned route. As thoroughly discussed in Section VI, the upper bounding procedure outperforms the heuristic proposed in [25], in those non-real-time settings where it is considered reasonable to wait half a minute to obtain high quality TDTSP solutions. Following the classification of [20], the algorithmic structure adopted in this contribution is refereed to as *learning to configure algorithms*, where machine learning is used to augment an operation research algorithm with valuable pieces of information. In particular, it is proposed an upper bounding technique inspired by the new findings of the recent paper [26], where the authors studied a property of time-dependent graphs, dubbed *path ranking invariance*. Given a time-dependent graph if the ordering of its paths (w.r.t. travel time) is independent of the start travel time, then the graph is *path ranking invariant*. The authors showed that, when a graph is path ranking invariant, a relevant class of time-dependent vehicle routing problems (with continuous piecewise travel times), including the TDTSP, can be solved by determining the optimal solution of their (simpler) time-independent counterpart. The authors demonstrated that the ranking invariance property can be checked by solving a (*large*) Linear Programming (LP) problem. If the ranking invariance check fails, they proved that a tight lower bound can be derived from the obtained LP solution.

This paper shows how the new findings of [26] can be further generalized for determining tight upper bounds for the TDTSP, with time-dependent travel times satisfying the FIFO property, but not (necessarily) continuous-piecewise linear. The main idea is to determine a heuristic solution by solving the TDTSP on an *auxiliary* time dependent graph, which satisfies the path ranking invariant property. The travel time functions of the auxiliary graph are determined by generalizing the LP-based approach proposed in [26]. In order to obtain a fast computation of the *auxiliary* travel time functions, the predictive component of a supervised ML technique has been exploited. Indeed, the ultimate goal is the fast computation of tight upper bounds, in those settings, customary in distribution management, in which *similar* instances are solved over and over again. As stated in [20], a *company does not care about solving all possible TSPs, but only theirs*. Therefore, instead of starting every time from scratch in the definition of the auxiliary graph, a learning mechanism has been inserted in such a way the upper bounding procedure can take advantage from previous runs on other (similar) instances. To this aim, the LP-based approach of [26] is boosted with a mix of supervised and unsupervised techniques.

The main contributions can be summarized as follows.

- An upper bounding procedure is proposed based on a *combinatorial* relaxation of the TDTSP, where time-dependent travel times satisfy the FIFO property, but are not (necessarily) continuous-piecewise linear.
- It is devised an automatic procedure for determining the parameters of the combinatorial relaxation, based on the combined use of a Linear Program and a mix of supervised and unsupervised Machine Learning techniques.
- It is generated a set of problem instances based on a real road network to show how the proposed heuristic approach can learn from past data to solve the TDTSP in an efficient and effective manner.

The paper is organized as follows. Section II provides a problem definition and some background information on the study area. Section III gives an overview of the whole solving method. Section IV introduces a parameterized family of upper bounds computed by solving the TDTSP on suitably defined *auxiliary* time-dependent graphs. Such family of upper bounds gives rise to an optimization problem aiming to determine the parameter providing the best (minimum) upper bounds. Section V proposes a ML-based heuristic approach for solving such optimization problem. Section VI discusses computational experiments on instances derived from the graphs of two European cities (London and Paris). Finally, Section VII draws some conclusions.

II. PROBLEM DEFINITION AND BACKGROUNDS

Let $[0, T]$ denote the time interval associated to a single working day. Without loss of generality it is supposed that the travel time functions are constant in the long run, that is $\tau_{ij}(t) := \tau_{ij}(T)$ with $t \geq T$. Furthermore, it is assumed that *first-in-first-out* (FIFO) property holds for the traversal time $\tau_{ij}(t)$, i.e., leaving the vertex i later implies arriving later at vertex j. For the sake of notational convenience, $\tau(i, j, t)$ is also used to designate $\tau_{ij}(t)$.

For any given path $p_k := (i_0, i_1, \ldots, i_k)$, the corresponding duration $z(p_k, t)$ can be computed recursively as:

$$z(p_k, t) := z(p_{k-1}, t) + \tau_{i_{k-1} i_k}(z(p_{k-1}, t)), \tag{1}$$

with the initialization $z(p_0, t) := t$. Therefore, a compact formulation of the TDTSP is:

$$\min_{p \in P} z(p, 0).$$

where P denotes the set of Hamiltonian tours on the time dependent graph $G := (V \cup \{0\}, A, \tau)$. Algorithms developed for the *classical* time-invariant TSP requires essential structural modifications in order to take into account time-varying travel times. Although time-dependent travel times have an impact on the ranking of solutions, they pose a difficulty for checking feasibility of solutions, only for those variants of TDTSP where it is required the fulfillment of time windows. Therefore, a quite natural way of defining a heuristic solution approach is to determine the optimal solution of a *classical* Asymmetric TSP (ATSP), defined on a graph $G_c = (V \cup \{0\}, A, c)$ where $c : A \rightarrow \mathbb{R}^+$ is a time-invariant (dummy) cost function. The main issue in this approach is how to determine a time-invariant (dummy) cost function that *mimics* in an effective manner the solutions ranking of the original TDTSP. In this respect, it can be proved that there always exists a time-invariant (dummy) cost function such that a least duration route of TDTSP is also a least cost solution of the TSP defined on the time-invariant graph G_c, which motivates the following definition.

Definition 1 (Valid Cost Function): A time-invariant cost function $c : A \rightarrow \mathbb{R}^+$ is *valid* for the TDTSP defined on $G = (V \cup \{0\}, A, \tau)$, if the least duration solution $p^* = \min_{p \in P} z(p, 0)$ corresponds to a least cost solution of the time-invariant ATSP defined on $G_c = (V \cup \{0\}, A, c)$, that is:

$$\arg\min_{p \in \mathcal{P}} \sum_{(i,j) \in \mathcal{P}} c(i, j) = \arg\min_{p \in \mathcal{P}} z(p).$$

Given a cost function *valid* for an instance of the TDTSP, the least duration solution p^* can be determined by exploiting algorithms developed for (*classical*) time invariant ATSP. In [26] the authors studied the relationship between the concept of *valid* cost function and a property of time-dependent graphs called *path ranking invariance*.

*Definition 2 (**Path ranking invariance**): A time-dependent graph G is path ranking invariant*, if for any pair of paths p' and p'' of G holds either:

$$z(p', t) \geq z(p'', t) \quad \forall t \geq 0,$$

or

$$z(p'', t) \geq z(p', t) \quad \forall t \geq 0.$$

Since travel time functions are constant in the long run, if a time-dependent graph $G = (V \cup \{0\}, A, \tau)$ is path ranking invariant then a *valid* cost function is $c(i, j) = \tau_{ij}(T)$.

A. THE AUXILIARY GRAPH

The proposed heuristic algorithm is based on the definition of an auxiliary path ranking invariant graph $\underline{G} = (V \cup \{0\}, A, \underline{\tau})$ where each $\underline{\tau}_{ij}(t)$ is an *approximation* of $\tau_{ij}(t)$, with $(i, j) \in A$. Each continuous piecewise linear function $\underline{\tau}_{ij}(t)$ is generated by the travel time model proposed in [27] (IGP model for short), in which each arc $(i, j) \in A$ is characterized by a constant stepwise speed function $v_{ij}(t)$ and a length L_{ij}. It is supposed that the horizon is partitioned into H subintervals $[T_h, T_{h+1}]$ $(h = 0, \ldots, H - 1)$, with $T_0 = 0$ and $T_H = T$. Furthermore, it is assumed that all arcs of the auxiliary graph $\underline{G}$ share a common speed function, such that

$$v_{ij}(t) = v_h,$$

with $t \in [T_h, T_{h+1}]$, $h = 0, \ldots, H-1$ and $(i, j) \in A$. According to the IGP model, given a start time t the travel time value $\underline{\tau}_{ij}(t)$ is computed by the following iterative procedure.

Algorithm 1 Computing the Travel Time $\underline{\tau}_{ij}(t)$

$q \leftarrow h : t_h \leq t \leq t_{h+1}$
$\ell \leftarrow L_{ij}$;
$t' \leftarrow t + \ell / v_q$;
while $t' > T_{q+1}$ **do**
  $\ell \leftarrow \ell - v_q(T_{q+1} - t)$;
  $t \leftarrow T_{q+1}$;
  $t' \leftarrow t + \ell / v_{q+1}$;
  $q \leftarrow q + 1$
return $t' - t$

In the IGP model the speed of a vehicle is not a constant over the entire length of arc $(i, j) \in A$ but it changes when the boundary between two consecutive time periods is crossed. Since the travel speed is a constant stepwise function, equality (2) represents a compact formulation of the relationship between the input parameters and the output value of the IGP model.

$$L_{ij} = \int_{t}^{t+\underline{\tau}_{ij}(t)} v(\mu) d\mu. \tag{2}$$

$\underline{z}(p_k, t)$ denotes the duration of a path p_k on the time-dependent graph $\underline{G}$, with t representing the start travel time, that is

$$\underline{z}(p_k, t) = \underline{z}(p_{k-1}, t) + \underline{\tau}_{i_{k-1} i_k}(\underline{z}(p_{k-1}, t)), \tag{3}$$

with the initialization $\underline{z}(p_0, t) = t$.

Proposition 1: ([26]) The time dependent graph $\underline{G} = (V \cup \{0\}, A, \underline{\tau})$ is path ranking invariant.

Proof: It is worth noting that from (2) it follows that given a path p it happens that:

$$\sum_{(i,j) \in p} L_{ij} = \int_{t}^{t+\underline{z}(p,t)} v(\mu) d\mu,$$

where the notation $(i, j) \in p$ means that the arc $(i, j) \in A$ is traversed by the path p. This implies that if a path p' is shorter than a path p'' then p' is also quicker than p'' for any start time $t \in [0, T]$:

$$\sum_{(i,j) \in p'} L_{ij} \leq \sum_{(i,j) \in p''} L_{ij} \Leftrightarrow \underline{z}(p', t) \leq \underline{z}(p'', t),$$

which proves the thesis. ■

The main implication of Proposition 1 is that an upper bound on the TDTSP defined on the *original* graph G can be obtained by solving a *classical* time invariant ATSP with cost coefficients $c(i, j) = \underline{\tau}_{ij}(T)$. Clearly the quality of the obtained upper bound is correlated with the fitting deviation between the original travel time function τ and its *approximation* $\underline{\tau}$. Minimizing such *fitting deviation* is the main idea underlying the family of parameterized upper bounds presented in the following sections.

III. PROBLEM-SOLVING METHOD

As illustrated in the previous section, given an instance of the TDTSP defined on G and the corresponding *valid* cost function, the optimal solution can be determined by solving a (classic) time-invariant ATSP. As stated in [26], the valid cost function is unknown and inaccessible except for path-ranking invariant graphs. The main goal is to construct an *approximator* of the valid cost function by combining machine learning and operations research (OR) algorithms, according to the *learning to configure* paradigm [20]. The basic underlying idea is to approximate the valid cost function with the valid cost function of an auxiliary (path-ranking invariant) graph. Algorithm 2 reports a general description of the proposed approach. The main components are an Artificial Neural Network (ANN), a Linear Program and an ATSP solver.

Artificial Neural Network. During a preprocessing step, the territory (and accordingly the customers) is partitioned in K zones using an unsupervised learning technique. A dataset of similar TDTSP instances (previously solved to optimality) is the training set of the ANN. Given the cardinalities of the set of customers for each zone, the ANN is trained to estimate the vector *ZETA* consisting of the mean expected arrival time at each zone in an optimal solution. Algorithm 2 receives as input the time-dependent graph G augmented with the coordinates of the K zones. The procedure starts with extracting from the time-dependent graph G the customer distribution n w.r.t. the set of K zones (Algorithm 2 - line 2). Then the ANN estimates the ZETA values of the TDTSP instance to be solved (Algorithm 2 - line 3). The estimated ZETAs are then exploited to determine the set Λ of time instants, then provided as input to the linear program (Algorithm 2 - line 4).

Linear Program. The approximated valid cost function $\underline{c}_\Lambda$ corresponds to the valid cost function of an auxiliary (time-dependent) graph $\underline{G}_\Lambda$, where the travel time functions are determined by solving the linear program $LP(G, \Lambda)$, i.e. the linear problem (7)-(14) defined on G and Λ (Algorithm 2 - line 5). The LP problem minimizes the expected fitting deviation between the original travel time functions τ and the auxiliary ones $\underline{\tau}_\Lambda$. In particular the fitting deviations refers to the set of time instants Λ generated from the neighbourhoods of the ZETA values determined by the ANN. The intuition is that, by taking a snapshot *around* the optimal arrival times (of similar instances previously solved), there is a good chance that the auxiliary graph *mimics* the arc ranking associated to the original (unknown and inaccessible) valid cost function.

ATSP solver. The heuristic solution $\underline{p}^*_\Lambda$ is determined by solving the TDTSP on the time-dependent (path-ranking invariant) graph $\underline{G}_\Lambda$. Therefore the sequence of customers $\underline{p}^*_\Lambda$ is determined by solving an ATSP instance with the same number of customers of the TDTSP instance, and the distance matrix filled with the values of the approximated valid cost function $\underline{c}_\Lambda$ (Algorithm 2 - lines 6-7).

The output of Algorithm 2 is the sequence of customers determined by the ATSP solver along with its duration w.r.t. the original travel time functions. Subsequent sections will provide all required insights following a bottom-up approach.

Algorithm 2 Problem-Solving Method

```
1: function Run(G)
2:     n ←Extract customer distribution of G
3:     ZETA ←ANN(n)
4:     Generate the set Λ from ZETA
5:     G_Λ ← Solve to optimality LP(G, Λ)
6:     c_Λ ← τ_Λ(T)
7:     p*_Λ ←Solve ATSP(c_Λ)
8:     z_Λ ← evaluate p*_Λ w.r.t. G
9:     return z_Λ, p*_Λ
```

IV. A FAMILY OF PARAMETERIZED UPPER BOUNDS

The bounding procedure is based on the *combinatorial* relaxations for TDTSP proposed in [26], where (original) travel times are required to be piecewise linear. This section discusses how such approach can be generalized to account for time-dependent travel times τ not (necessarily) continuous piecewise linear. To this aim it is defined a family of parameterized upper bounds $\underline{z}_\Lambda$, where parameters Λ constitute an ordered set of time instants. Given set Λ, upper bound $\underline{z}_\Lambda$ is determined by solving the TDTSP on an auxiliary path ranking invariant graph $\underline{G}_\Lambda = (V, A, \underline{\tau}_\Lambda)$. The travel time function $\underline{\tau}_\Lambda$ is an approximation of the original travel function τ. In particular $\underline{\tau}_\Lambda$ is generated by the IGP model and satisfies relationship (2). Recall that the IGP parameters are: the set of speed breakpoints, the speed values and the length of the arcs. The given *upper-bound parameter* Λ is used to model the set of IGP speed breakpoints, i.e. $\Lambda = \{T_0, \ldots, T_H\}$, with $H = |\Lambda| - 1$ (Algorithm 2 - line 4). Then speed values and length of arcs are prescribed by a linear program, which aims to minimize *the fitting deviation* between the original τ and its parameterized approximation $\underline{\tau}_\Lambda$ (Algorithm 2 - line 5). The main idea underlying the linear program is that the equalities (2) imply that the travel time functions τ and $\underline{\tau}_\Lambda$ are perfect fit if the following relationship holds for each arc $(i, j) \in A$ and time instant $t \in T$:

$$L_{ij} - \int_t^{t+\tau_{ij}(t)} v(\mu)d\mu = 0. \tag{4}$$

The objective function aims to minimize a *fitting deviation* given by the violations of equality constraints (4). Due to the continuous time nature of (4), a surrogate of the *fitting deviation* is defined by evaluating (4) only for time instants belonging to a set Λ_{ij}, that is:

$$L_{ij} - \int_{T_h}^{T_h+\tau_{ij}(T_h)} v(\mu)d\mu = 0, \tag{5}$$

with $h = 0, \ldots, |\Lambda_{ij}| - 1$ and $(i, j) \in A$. The set Λ is defined as the union set of Λ_{ij}, with $(i, j) \in A$, i.e. $\Lambda = \bigcup_{(i,j)\in A} \Lambda_{ij}$.

Let a_{ijkh} define the coefficient representing time spent on arc (i, j) during period h when departing at T_k, that is:

$$a_{ijkh} = \begin{cases} \min(T_{h+1} - T_h, \max(0, T_k + \tau_{ij}(T_k) - T_h)) & k \leq h \\ 0 & otherwise \end{cases}$$

with $(i, j) \in A$, $h, k = 0, \ldots, |\Lambda_{ij}| - 1$.

Since $v(t)$ is constant stepwise, relationship (5) can be expressed by the following linear equality:

$$\sum_{h=0}^{|\Lambda_{ij}|-1} a_{ijkh} \cdot v_h = L_{ij} + s_{ijk}, \tag{6}$$

where the *free-sign* variable s_{ijk} models the *violation* of the right-hand-side of (5) with respect to L_{ij}, with $(i, j) \in A$, $k = 0, \ldots, |\Lambda_{ij}| - 1$. The proposed linear program determines a speed function $v(t)$ and the corresponding right-hand-sides of (6), which is denoted with x_{ijk}: since it represents a length it is required that $x_{ijk} \geq 0$, with $(i, j) \in A$, $k = 0, \ldots, |\Lambda_{ij}| - 1$. The *maximum fitting deviation* between the original travel time function $\tau(i, j, t)$ and $\underline{\tau}_\Lambda(i, j, t)$ is modelled as

$$\zeta_{ij} = \max_{k\in[0,\ldots,|\Lambda_{ij}|-1]} x_{ijk} - \min_{k\in[0,\ldots,|\Lambda_{ij}|-1]} x_{ijk},$$

with $(i, j) \in A$. Quantity $\zeta_\Lambda = \sum_{(i,j)\in A} \zeta_{ij}$ represents an *approximated* measure of the *total fitting deviation* associated to the auxiliary graph $\underline{G}_\Lambda$. The auxiliary graph $\underline{G}_\Lambda$ is determined in such a way that the corresponding travel time function $\underline{\tau}_\Lambda$ minimizes the value of ζ_Λ. To this aim, it is formulated the following linear program (7)-(14), where $\underline{x}_{ij}$ and $\overline{x}_{ij}$ model, respectively, the minimum and maximum value of the variables x_{ijk}, with $(i,j) \in A$ and $k = 0, \ldots, |\Lambda_{ij}| - 1$. A solution of such linear programming model also prescribes the parameters of a stepwise function $y(t)$. In particular,

$$y(t) = y_h,$$

that is during the $h-th$ time interval $y(t)$ assumes the value prescribed by the continuous variable y_h, with $t \in [t_h, t_{h+1}]$ and $h = 0, \ldots, |\Lambda| - 1$.

$$\zeta_\Lambda^* := \min \sum_{(i,j)\in A} \overline{x}_{ij} - \underline{x}_{ij} \tag{7}$$

s.t.

$$\sum_{h=0}^{|\Lambda_{ij}|-1} a_{ijkh} \cdot y_h = x_{ijk} \qquad k = 0, \ldots, |\Lambda_{ij}| - 1 \quad (i,j) \in A \tag{8}$$

$$\underline{x}_{ij} \leq x_{ijk} \qquad k = 0, \ldots, |\Lambda_{ij}| - 1, (i,j) \in A \tag{9}$$

$$\overline{x}_{ij} \geq x_{ijk} \qquad k = 0, \ldots, |\Lambda_{ij}| - 1, (i,j) \in A \tag{10}$$

$$x_{ijk} \geq 0, \qquad k = 0, \ldots, |\Lambda_{ij}| - 1, (i,j) \in A \tag{11}$$

$$\underline{x}_{ij} \geq 0 \qquad (i,j) \in A \tag{12}$$

$$\overline{x}_{ij} \geq 0 \qquad (i,j) \in A \tag{13}$$

$$y_h \geq \rho \qquad h = 0, \ldots, |\Lambda| - 1 \tag{14}$$

Objective function (7) aims to determine a step function $y^*(t)$ that minimizes the total maximum fitting deviation between the original travel time function τ and its approximation $\underline{\tau}_\Lambda$. Constraints (8) state the relationship between $y(t)$ and x variables. Constraints (9) and (10) model the relationship between $\underline{x}_{ij}$, $\overline{x}_{ij}$ and continuous variables x_{ijk}. Constraints (11), (12), (13) and (14) describe the non-negative conditions on the decision variables. In particular, constraints (14) cut off the trivial (pointless) solution $y(t) = 0$ for $t \geq 0$. Let $y^*(t)$ and x^* denote, respectively, the step function and the x values associated with the optimal solution of the linear program (7)-(14). Moreover, $\tilde{x}^*_{ij}$ denotes the average of the x values associated to arc $(i,j) \in A$ in the optimal solution, that is:

$$\tilde{x}^*_{ij} = \sum_{h=0}^{|\Lambda_{ij}|-1} \frac{x^*_{ijh}}{|\Lambda_{ij}|}.$$

It is observed that the linear program does not *directly* prescribe the IGP parameter L_{ij}, with $(i,j) \in A$. Indeed, according to (6) it follows that:

$$x^*_{ijk} = L_{ij} + s_{ijk},$$

where, recal that, s_{ijk} quantifies the violation of equality (5), with $(i,j) \in A$ and $k = 0, \ldots, |\Lambda_{ij}| - 1$. Since L_{ij} denotes the IGP length associated with $\underline{\tau}_\Lambda$, from (6) it follows that

$$\int_{t_k}^{t_k+\tau(i,j,t_k)} v(\mu)d\mu - \int_{t_k}^{t_k+\underline{\tau}_\Lambda(i,j,t_k)} v(\mu)d\mu = s_{ijk},$$

that is the lower the *absolute* value of equality (5) *violation* (i.e. $|s_{ijk}|$), the lower the *absolute error* made by approximating $\tau(i,j,t_k)$ with $\underline{\tau}_\Lambda(i,j,t_k)$, with $t_k \in \Lambda_{ij}$ and $(i,j) \in A$. Since $\tilde{x}^*_{ij}$ minimizes the mean squared *violation* of equality (5), i.e.

$$\tilde{x}^*_{ij} = \arg\min_{L_{ij}} \sum_{k=0}^{|\Lambda_{ij}|-1} \frac{(x^*_{ijk} - L_{ij})^2}{|\Lambda_{ij}|},$$

such travel time *approximation* errors are (*heuristically*) minimized by generating the travel time function $\underline{\tau}_\Lambda(i,j,t)$ with the following IGP input parameters:

$$v(t) = y^*(t), \qquad L_{ij} = \tilde{x}^*_{ij},$$

with $(i,j) \in A$. Finally, remind that the travel time function $\underline{\tau}_\Lambda(i,j,t)$ satisfies relationship (2), and, therefore, the auxiliary graph is path ranking invariant. Summing up, given a set of time instants $\Lambda = \bigcup_{(i,j)\in A} \Lambda_{ij}$ and a time dependent graph G, the proposed upper bounding procedure is made up three main steps.

- **STEP 1**. Compute the optimal solution of the linear program (7)-(14). Set the travel speed function $v(t)$ equal to $y^*(t)$. Similarly set L_{ij} to $\tilde{x}^*_{ij}$ for each $(i,j) \in A$ (Algorithm 2 - line 5).
- **STEP 2**. Determine the optimal solution $\underline{p}^*_\Lambda$ of the following time-independent ATSP (Algorithm 2 - lines 6-7):

$$\min_{p\in\mathcal{P}} \sum_{(i,j)\in p} \underline{\tau}_\Lambda(i,j,T).$$

- **STEP 3**. Determine upper bound $\underline{z}_\Lambda$ as the duration of $\underline{p}^*_\Lambda$ evaluated w.r.t. the original travel time function τ that is (Algorithm 2 - line 8):

$$\underline{z}_\Lambda = z(\underline{p}^*_\Lambda, 0)$$

Finally, it is worth noting that in order to find the least upper bound, the following optimization problem has to be solved:

$$\min_\Lambda \underline{z}_\Lambda, \tag{15}$$

where $\underline{z}_\Lambda$ is evaluated according to the proposed three-steps procedure. A *simple* heuristic for solving (15) is to set each Λ_{ij} equal to a discretization $\mathcal{D}$ of the planning horizon. In this case, the three-steps procedure computing the upper bound $\underline{z}_\mathcal{D}$ is referred as PL-enhanced heuristic (PL-HTSP for short). The main drawback of the PL-HTSP heuristic is that the computation of a tight upper bound value $\underline{z}_\mathcal{D}$ might require the solution of a *large* Linear Program. Next section shows a machine learning based heuristic for solving (15) aiming to overcome this drawback. In particular, the predictive capabilities of machine learning is exploited in order to carefully select Λ as a (quite small) subset of time instants in $\mathcal{D}$. In this case, the three-steps upper bounding procedure computing $\underline{z}_\Lambda$ is referred to as MLPL-enhanced heuristic (MLPL-HTSP for short).

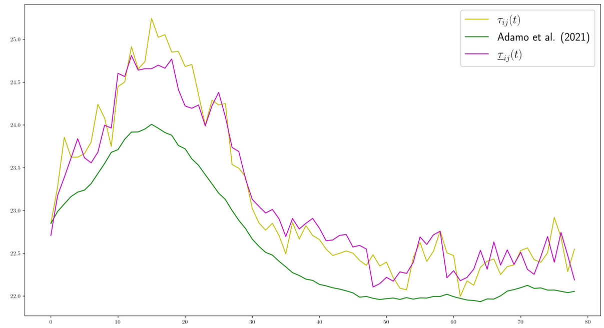

FIGURE 1. **Comparing the $\underline{\tau}$ functions determined by, respectively, the approximation procedure and [26].**

V. LEARNING TO ENHANCE UPPER BOUNDS

This section proposes a learning mechanism for determining set Λ (deepening the above Algorithm 2 - line 4). Then upper bound $\underline{z}_\Lambda$ is computed according to the three-steps upper bounding procedure illustrated in the previous section. As stated in Section I, the goal is to determine "good" upper bounds, by reusing the information gained when solving instances with similar features. To this aim, instead of starting every time from scratch in the definition of the auxiliary graph $\underline{G}_\Lambda$, a learning mechanism is devised so that the upper bounding procedure can benefit from previous runs on other instances with similar features.

The idea of *bounds based on an auxiliary path ranking invariant graph* is inspired by [26], where the authors devised a family of parameterized *combinatorial* relaxations for the TDTSP. They proposed a procedure to determine *auxiliary* travel times which are "good" lower approximations of the original ones. Then a lower (*dual*) bound is determined by solving the TDTSP on the *less congested* auxiliary graph. This research work is aimed to devise a procedure for determining an upper (*primal*) bound by solving a TDTSP on an auxiliary path ranking invariant graph. As shown in the example reported in Fig. 1, by applying this approach, the aim is to get a travel time approximation that fits the original τ better than the lower approximation determined by [26]. In particular this paper proposes a mechanism for *learning* the relationship between set Λ and the optimal solutions of the TDTSP defined on the original time-dependent graph G. First of all, it is observed that there exists a finite and discrete set Λ^*, consisting of all (*feasible*) arrival times: if t belongs to Λ^*, then there exists on G a feasible tour $p \in P$ with t corresponding to the arrival time at a node $i \in V$. That such set Λ^* exists is based on the observation that there is a finite number of feasible tours.

Remark 1: If $\zeta^*_{\Lambda^*} = 0$, then for each arc $(i,j) \in A$ and time instants $t \in \Lambda^*$, it follows that:

$$\underline{\tau}_{\Lambda^*}(i,j,t) = \tau(i,j,t)$$

and therefore, upper bound $\underline{z}_{\Lambda^*}$ is optimal, that is $\underline{z}_{\Lambda^*} = \min_{p \in P} z(p,0)$.

The main limit of the sufficient optimality condition stated in Remark 1 is that determining the entire Λ^* is computationally challenging. To overcome this drawback, the predictive capabilities of supervised ML techniques have been exploited, in order to determine a set Λ such that the arrival times associated to optimal solutions have a good chance of being included in Λ. Let f_i denote a prediction (obtained through a supervised ML method) of the *expected time of arrival* (ETA) at customer i in an optimal solution. It is observed that the ranking among arcs might deeply change during the planning horizon on the original graph G. On the other hand, the path ranking invariance of the auxiliary graph $\underline{G}_\Lambda$ holds for any pair of paths, each one consisting of at least one arc. This also implies an *arc* ranking invariance on $\underline{G}_\Lambda$. The intuition is that, by taking a snapshot *around* the optimal arrival times (of similar instances previously solved), there is a good chance of embedding in the auxiliary graph $\underline{G}_\Lambda$ the arc ranking associated to the set of quickest tours of the original graph. For this purpose, the maximum fitting deviation between the original travel time function $\tau(i,j,t)$ and $\underline{\tau}_\Lambda(i,j,t)$ is minimized for each arc $(i,j) \in A$ in the time interval $[f_i - \epsilon_i, f_i + \epsilon_i]$, where $\epsilon_i > 0$ represents the mean absolute error associated to f_i, with $i \in V$.

In particular, let $\mathcal{D}$ define a discretization of the time horizon. Then for each node i, a subset S_i of $\mathcal{D}$ is selected as follows:

$$S_i = \{t \in [f_i - \epsilon_i, f_i + \epsilon_i] \land t \in \mathcal{D}\}$$

In the definition of the approximation travel time $\underline{\tau}_\Lambda$, all arcs $(i,j) \in A$ outgoing the node $i \in V$ share a common set Λ_{ij} corresponding to the set S_i, i.e. $\Lambda_{ij} = S_i$. Therefore in the MLPL-HTSP, the travel time $\underline{\tau}_\Lambda$ is determined by solving the linear program (7)-(14), where the role of Λ_{ij} is played by the subset S_i in the constraints (8)-(11), with $i = 1, \ldots, n$.

A. ETA ESTIMATION

Given a training instance, the exact algorithm devised by [10] has been used in order to obtain the optimal arrival times at the customers. The estimation of ETA for each customer i in an optimal solution for a new instance has been obtained through an artificial neural network (ANN). *Multilayer Perceptron Regressor* (MPR) is the chosen ANN reference implementation with at least three layers: one layer composed by input nodes, one or more for hidden nodes and one for output nodes ([28]). A nonlinear activation function was used by all nodes not belonging to the input layer. The ANN has K nodes in the input layer and K nodes in the output layer. K also represents the number of zones in which the territory (and accordingly customers) has been partitioned using an unsupervised learning technique. In the computational campaign, K-means algorithm has been used to aggregate the customers belonging to instances of the training set into K clusters which minimizes within-cluster variances ([29]). ANN inputs are the number n_k $k = 1, \ldots, K$ of customers in each zone (namely, the customers distribution as pointed out in Algorithm 2 - line 2); whilst ANN outputs are $ZETA_k$

TABLE 1. ANN mean errors on the London instances.

Zone	Mean error	Mean absolute error	Standard error
1	7.68	36.78	55.16
2	-4.61	29.23	37.19
3	8.32	26.94	35.51
4	-1.93	27.34	36.87
5	-2.68	28.78	46.21
6	8.69	56.68	69.21
7	2.54	24.60	32.31
8	6.68	54.00	64.84
Average	3.09	35.54	47.16

TABLE 2. ANN mean errors on Paris instances.

Zone	Mean error	Mean absolute error	Standard error
1	-1.02	18.55	23.74
2	2.40	15.29	20.14
3	0.74	19.69	24.30
4	-2.78	28.85	36.53
5	5.53	44.65	52.49
6	1.33	24.00	29.55
Average	1.03	25.17	31.13

$k = 1, \ldots, K$ the mean zone ETA (Algorithm 2 - line 3). Lower K implies a high variability of ETA values in a zone. In contrast, larger K corresponds to more accurate predictions, but the training set should be very huge. A preliminary experimentation allowed the definition of an optimal K value.

VI. COMPUTATIONAL EXPERIMENTS

The quality of the proposed upper bounding procedure was empirically assessed through a computational campaign.

The branch-and-bound scheme proposed in [10] enhanced with the lower bound proposed in [26] has been used to solve every training instance, imposing a time limit of an hour. The Asymmetric TSP subproblems have been solved by means of [30]. The linear program (7)-(14) was solved with IBM ILOG CPLEX 12.10. The machine learning component of the MLPL-HTSP algorithm was implemented in Python (version 3.7). The MPR and K-means implementations were taken from *scikit-learn* machine learning library. All experimentation have been conducted on a Linux machine with 4 cores at 2.67 GHz and 8 GB of RAM installed. Instances are based on the real travel time functions of Paris and London [25] (available at `https://tdrouting.com/instances.zip`).

A. PARAMETER TUNING

A preliminary tuning phase permitted to select the most appropriate combination of parameters. The Paris dataset is composed by 600 instances, whereas London one counts 700 instances; all instances have 50 customers each. For both cities, the full dataset has been splitted into a training set composed by 90% of the instances, and a validation set with the remaining 10%. The ANN with the best performance in terms of strength of caught interconnections has the following parameters: hyperbolic-tangent as activation function, five neurons in a single hidden layer, LBFGS optimizer with constant learning rate. With respect to customer partitioning, Table 1 and Table 2 reports the ANN mean errors (in minutes) for each zone. In particular, the best results in terms of coefficient of determination (R^2) have been obtained considering 8 clusters for the London instances and 6 zones for the Paris instances. It is worth noting that the R^2 score ($= 0.53$ for London and $= 0.60$ for Paris) indicates a medium effect size. Parameter ϵ_i has been set equal to the mean absolute error of the zone, which the customer $i \in V$ belongs to. A 5-minutes time unit has been considered for the discretization $\mathcal{D}$ of the planning horizon. Finally, ρ has been set equal to $1/\min_{h=0,\ldots,|\Lambda|-1}(T_{h+1} - T_h)$.

TABLE 3. Impact of approximation $\underline{\tau}$ and the machine learning algorithm on solution quality.

Testset	Heuristic	Avg DEV%	min DEV	max DEV
London	HTSP	1.42%	0.00	16.44
London	PL-HTSP	0.35%	-0.90	8.36
London	MLPL-HTSP	0.23%	-0.49	8.16
Paris	HTSP	0.72%	-9.45	11.04
Paris	PL-HTSP	-0.14%	-13.13	13.13
Paris	MLPL-HTSP	-0.18%	-12.52	3.93

TABLE 4. Impact of approximation $\underline{\tau}$ and the machine learning algorithm on computing time.

Testset	Heuristic	Avg Time	min Time	max Time
London	HTSP	1.26	0.08	7.18
London	PL-HTSP	128.52	91.72	195.34
London	MLPL-HTSP	18.28	14.95	26.40
Paris	HTSP	1.94	0.06	10.90
Paris	PL-HTSP	83.12	57.11	105.93
Paris	MLPL-HTSP	12.46	8.73	37.47

B. COMPUTATIONAL RESULTS

As illustrated in the previous section, the predictive capabilities of the ML-techniques have been exploited for the fast computation of two Λ sets, associated to London and Paris respectively. Then the two testsets were solved by the MLPL-HTSP algorithm. The computational results are presented in Tables 5 - Table 6, under the following headings:

- the name of the test instance,
- the objective value BK in minutes of the best-known solution determined by the exact algorithm proposed in [10] enhanced with the lower bound proposed in [26], with a time limit of 1 hour;
- the objective value $\underline{z}_\Lambda$ in minutes of the MLPL-HTSP solution;
- the deviation DEV of $\underline{z}_\Lambda$ w.r.t. BK in percentage, computed as:

$$DEV = \frac{\underline{z}_\Lambda - BK}{BK};$$

- *Time* in seconds spent to determine $\underline{z}_\Lambda$.

The new best-known solution for $\underline{z}_\Lambda$ are shown in bold. The average running times are 18.28 seconds for the London instances and 12.46 seconds for Paris instances. The average percentage deviation between MLPL-HTSP result and the best-known solution is 0.23% for the London instances and -0.18% for the Paris instances. In the worst case, the percentage deviation is 2.15% and in 31 cases a new best-known

TABLE 5. MLPL-HTSP results on London test instances.

Instance	BK	$\underline{z}_\Lambda$	$DEV\%$	time
10_I_1	407.59	407.59	0.00%	17.90
10_I_10	379.27	387.43	2.15%	18.44
10_I_11	400.62	403.28	0.66%	21.28
10_I_12	401.17	402.09	0.23%	19.73
10_I_13	463.42	463.42	0.00%	24.86
10_I_14	399.75	399.77	0.01%	21.40
10_I_15	415.50	418.84	0.80%	18.34
10_I_16	401.62	401.81	0.05%	16.84
10_I_17	402.36	402.36	0.00%	15.60
10_I_19	436.13	436.13	0.00%	18.80
10_I_2	372.64	**372.31**	-0.09%	15.25
10_I_20	422.78	425.09	0.55%	17.53
10_I_23	400.79	400.82	0.01%	18.75
10_I_24	411.51	413.28	0.43%	18.93
10_I_25	404.39	404.64	0.06%	17.52
10_I_26	409.90	410.32	0.10%	18.72
10_I_27	420.02	420.02	0.00%	19.97
10_I_28	419.80	421.90	0.50%	19.94
10_I_29	408.59	409.82	0.30%	20.94
10_I_30	395.66	396.32	0.17%	15.70
10_I_31	409.23	411.73	0.61%	24.82
10_I_32	398.56	**398.07**	-0.12%	15.58
10_I_33	345.61	350.94	1.54%	17.12
10_I_34	353.48	353.52	0.01%	18.41
10_I_36	394.61	394.61	0.00%	15.91
10_I_37	416.03	416.59	0.13%	16.02
10_I_38	453.65	453.79	0.03%	19.90
10_I_39	426.38	426.49	0.03%	17.30
10_I_40	416.32	417.37	0.25%	18.13
10_I_41	398.48	398.48	0.00%	16.61
10_I_5	393.85	395.13	0.32%	19.25
10_I_6	399.36	399.36	0.00%	15.70
10_I_7	388.38	388.69	0.08%	19.75
10_I_9	369.03	369.79	0.21%	18.84
1_I_2	388.70	390.75	0.53%	18.53
1_I_26	419.04	419.04	0.00%	16.68
1_I_27	378.45	378.45	0.00%	16.43
1_I_28	393.14	394.52	0.35%	16.37
1_I_29	393.51	394.14	0.16%	23.73
1_I_3	396.82	399.36	0.64%	15.48
1_I_30	387.16	387.16	0.00%	15.33
1_I_31	363.90	363.90	0.00%	14.95
1_I_32	408.21	408.21	0.00%	17.31
1_I_33	414.32	415.26	0.23%	21.69
1_I_34	365.65	365.94	0.08%	15.65
1_I_35	412.53	412.53	0.00%	19.08
1_I_36	369.79	374.14	1.18%	19.30
1_I_37	410.90	410.91	0.00%	16.71
1_I_39	406.39	407.94	0.38%	22.43
1_I_4	402.54	402.65	0.03%	26.40
1_I_40	396.62	396.62	0.00%	15.03
1_I_42	408.81	408.81	0.00%	20.21
1_I_44	373.48	374.71	0.33%	21.97
1_I_45	367.21	367.26	0.01%	15.16
1_I_46	404.26	404.59	0.08%	17.55
1_I_47	402.02	402.61	0.15%	18.54
1_I_48	393.13	394.97	0.47%	16.31
1_I_49	381.64	381.64	0.00%	16.16
1_I_5	333.64	335.85	0.66%	15.96
1_I_50	372.23	372.62	0.10%	16.18
1_I_51	417.30	417.74	0.11%	18.47
1_I_53	405.22	405.22	0.00%	16.08

solution is found. For 38 instances, the MLPL-HTSP heuristic also obtains the best known solution, whilst for 100 out of 140 instances the absolute value $|BK - \underline{z}_\Lambda|$ is less or equal than 1 minute, which is the smallest time unit meaningful in real vehicle routing problems inside large cities.

The impact of both the linear program (7)-(14) and the machine learning algorithm have been also examined. To this end, a baseline heuristic HTSP has been devised, where the

TABLE 6. MLPL-HTSP results on Paris test instances.

Instance	BK	$\underline{z}_\Lambda$	$DEV\%$	time
0_I_0	289.26	289.26	0.00%	15.59
0_I_1	282.15	282.23	0.03%	11.54
0_I_10	291.04	291.09	0.02%	9.98
0_I_100	285.31	285.31	0.00%	10.04
0_I_101	286.66	**274.14**	-4.37%	18.64
0_I_102	273.71	273.88	0.06%	9.55
0_I_103	297.27	297.27	0.00%	11.36
0_I_104	289.87	290.07	0.07%	9.83
0_I_105	309.26	309.40	0.05%	9.75
0_I_106	286.73	286.82	0.03%	9.45
0_I_107	295.62	295.91	0.10%	10.71
0_I_108	279.18	**278.58**	-0.21%	9.67
0_I_109	287.85	287.85	0.00%	15.48
0_I_11	310.77	310.77	0.00%	11.61
0_I_110	274.52	278.46	1.44%	10.69
0_I_111	301.50	**300.51**	-0.33%	14.62
0_I_112	306.67	**305.80**	-0.28%	15.94
0_I_113	303.81	306.41	0.86%	14.32
0_I_114	298.17	**296.57**	-0.54%	14.69
0_I_115	293.19	294.04	0.29%	10.71
0_I_116	288.90	288.90	0.00%	24.52
0_I_117	300.82	**297.73**	-1.03%	10.92
0_I_118	275.94	275.98	0.01%	10.36
0_I_119	274.69	274.69	0.00%	9.65
0_I_12	301.23	302.65	0.47%	12.61
0_I_120	295.00	295.08	0.03%	11.40
0_I_121	289.19	289.31	0.04%	10.39
0_I_122	283.25	**281.89**	-0.48%	17.13
0_I_123	312.11	312.12	0.00%	11.90
0_I_124	300.24	**298.42**	-0.61%	14.63
0_I_125	285.50	285.64	0.05%	9.42
0_I_126	296.42	297.22	0.27%	21.34
0_I_127	299.22	299.25	0.01%	10.28
0_I_128	285.49	285.64	0.05%	15.82
0_I_129	282.04	282.04	0.00%	9.48
0_I_13	287.11	287.11	0.00%	13.10
0_I_130	315.47	**314.00**	-0.47%	15.41
0_I_131	271.56	271.57	0.00%	9.94
0_I_132	259.81	**259.75**	-0.02%	14.29
0_I_133	280.81	280.81	0.00%	11.74
0_I_134	287.89	288.53	0.22%	37.47
0_I_135	305.73	**304.78**	-0.31%	11.78
0_I_136	283.76	**283.43**	-0.12%	10.25
0_I_137	279.85	**279.47**	-0.14%	10.86
0_I_138	275.06	275.06	0.00%	9.10
0_I_139	300.82	**300.41**	-0.14%	11.04
0_I_14	277.91	**274.31**	-1.30%	9.31
0_I_140	295.50	**294.39**	-0.38%	10.12
0_I_141	300.23	**298.68**	-0.52%	13.27
0_I_142	285.36	**281.75**	-1.27%	11.91
0_I_143	287.65	287.65	0.00%	10.39
0_I_144	277.19	**276.35**	-0.30%	9.32
0_I_145	254.78	255.08	0.12%	8.79
0_I_146	288.52	288.62	0.03%	12.47
0_I_147	295.02	**292.48**	-0.86%	11.59
0_I_148	276.02	276.24	0.08%	8.73
0_I_149	289.43	289.69	0.09%	9.88
0_I_15	299.90	299.90	0.00%	10.92
0_I_150	290.86	**289.40**	-0.50%	11.92
0_I_151	283.60	283.77	0.06%	11.90
0_I_152	293.53	**287.85**	-1.94%	10.88
0_I_153	273.22	273.22	0.00%	10.88
0_I_154	289.59	**288.51**	-0.37%	10.20
0_I_155	318.15	318.15	0.00%	10.41
0_I_156	278.43	278.69	0.09%	9.34
0_I_157	292.37	**288.54**	-1.31%	11.95
0_I_159	292.76	294.04	0.44%	12.60
0_I_16	304.56	**301.95**	-0.86%	20.07
0_I_160	281.17	281.40	0.08%	12.63
0_I_161	305.14	305.14	0.00%	11.13
0_I_162	335.01	**334.37**	-0.19%	10.93
0_I_163	289.14	**287.52**	-0.56%	15.06
0_I_164	272.99	**272.87**	-0.04%	10.77
0_I_165	290.55	290.73	0.06%	10.09
0_I_166	308.36	308.57	0.07%	12.85
0_I_168	304.05	304.05	0.00%	9.78
0_I_169	280.77	280.90	0.05%	9.83
0_I_17	309.58	**309.05**	-0.17%	22.79

auxiliary graph $\underline{G}$ is time-independent, with the constant value associated to each arc $(i, j) \in A$ set equal to $\max_{t \in [0,T]} \tau_{ij}$, for each $(i, j) \in A$. Table 3 and Table 4 report results for all three heuristics: column headings are self explanatory. Results associated to the PL-HTSP highlight that the computation of the approximation $\underline{\tau}_\Lambda$ provides a remarkable increase of both the solution quality and the computing time w.r.t. the baseline heuristic HTSP. It is by leveraging the machine learning that the MLPL-HTSP heuristic obtains both solution quality improvement and a reduction (by an order of magnitude) of the computing time w.r.t. the PL-HTSP heuristic. Moreover it is observed that the MLPL-HTSP heuristic provides remarkable improvements in terms of both worst case and best case, i.e. the maximum and minimum values of DEV in Table 3. As far as the computing time is concerned, Table 4 shows that MLPL-HTSP represents a good tradeoff between the baseline algorithm and the PL-HTSP. Indeed, the maximum computing time of MLPL-HTSP is remarkably lower than the minimum time of PL-HTSP, whilst the minimum computing time of MLPL-HTSP is only few seconds above the maximum time of HTSP. It is worth noting that the upper bounding procedure consistently outperforms the ML heuristic proposed in [25], by providing an average saving on route duration equal to 49 minutes for the London instances and 30 minutes for the Paris instances.

The provided results clearly show which high quality performance are achieved by the MLPL-HTSP algorithm for instances that correspond to realistic travel time functions.

VII. CONCLUSION

The main contribution of this paper is an algorithm that learns from past data to solve the TDTSP in an efficient and effective manner. Computational results on two European cities show that the average gap with the best-known solutions is only 0.001% and the average computation time is 15 seconds. Furthermore, new best solutions have been produced for several test instances. This is achieved by solving a time-invariant Asymmetric TSP, where the arc (constant) costs are properly defined by the combined use of an LP-based approach and a mix of unsupervised and supervised ML techniques. In particular, a feedforward ANN has been trained on past instances solved to (near-)optimality, and its ETA predictions have been exploited. Future research could investigate the definition of new features for the neural network as well as exploit the use of deep learning methods [31]. Another noteworthy research goal concerns the study of a more efficient algorithm for (approximately) minimizing the fitting deviation between the travel time function τ and its *approximation* $\underline{\tau}_\Lambda$. Finally, future attempts could be aimed at the adaptation of the ideas introduced in this paper to other routing problems.

REFERENCES

[1] Y. Xiao, Y. Zhang, I. Kaku, R. Kang, and X. Pan, "Electric vehicle routing problem: A systematic review and a new comprehensive model with nonlinear energy recharging and consumption," *Renew. Sustain. Energy Rev.*, vol. 151, Nov. 2021, Art. no. 111567.

[2] Y. Xiao, X. Zuo, J. Huang, A. Konak, and Y. Xu, "The continuous pollution routing problem," *Appl. Math. Comput.*, vol. 387, Dec. 2020, Art. no. 125072.

[3] M. Gendreau, G. Ghiani, and E. Guerriero, "Time-dependent routing problems: A review," *Comput. Oper. Res.*, vol. 64, pp. 189–197, Dec. 2015.

[4] C. Malandraki and M. S. Daskin, "Time dependent vehicle routing problems: Formulations, properties and heuristic algorithms," *Transp. Sci.*, vol. 26, no. 3, pp. 185–200, 1992.

[5] C. Malandraki and R. B. Dial, "A restricted dynamic programming heuristic algorithm for the time dependent traveling salesman problem," *Eur. J. Oper. Res.*, vol. 90, no. 1, pp. 45–55, 1996.

[6] F. Li, B. Golden, and E. Wasil, "Solving the time dependent traveling salesman problem," in *The Next Wave in Computing, Optimization, and Decision Technologies* (Operations Research/Computer Science Interfaces Series), vol. 29, R. Sharda, S. Voß, B. Golden, S. Raghavan, and E. Wasil, Eds. Berlin, Germany: Springer, 2005, pp. 163–182.

[7] J. Schneider, "The time-dependent traveling salesman problem," *Phys. A, Stat. Mech. Appl.*, vol. 314, pp. 151–155, Nov. 2002.

[8] K. Harwood, C. Mumford, and R. Eglese, "Investigating the use of metaheuristics for solving single vehicle routing problems with time-varying traversal costs," *J. Oper. Res. Soc.*, vol. 64, no. 1, pp. 34–47, Jan. 2013.

[9] J.-F. Cordeau, G. Ghiani, and E. Guerriero, "Analysis and branch-and-cut algorithm for the time-dependent travelling salesman problem," *Transp. Sci.*, vol. 48, no. 1, pp. 46–58, Feb. 2014.

[10] A. Arigliano, T. Calogiuri, G. Ghiani, and E. Guerriero, "A branch-and-bound algorithm for the time-dependent travelling salesman problem," *Networks*, vol. 72, no. 3, pp. 382–392, Oct. 2018.

[11] P. A. Melgarejo, P. Laborie, and C. Solnon, "A time-dependent no-overlap constraint: Application to urban delivery problems," in *Proc. Int. Conf. AI Techn. Constriant Program. Combinat. Optim. Problems*. Cham, Switzerland: Springer, 2015, pp. 1–17.

[12] T. Adamo, G. Ghiani, and E. Guerriero, "An enhanced lower bound for the time-dependent travelling salesman problem," *Comput. Oper. Res.*, vol. 113, Jan. 2020, Art. no. 104795.

[13] J. Albiach, J. M. Sanchis, and D. Soler, "An asymmetric TSP with time Windows and with time-dependent travel times and costs: An exact solution through a graph transformation," *Eur. J. Oper. Res.*, vol. 189, no. 3, pp. 789–802, Sep. 2008.

[14] A. Arigliano, G. Ghiani, A. Grieco, E. Guerriero, and I. Plana, "Time-dependent asymmetric traveling salesman problem with time windows: Properties and an exact algorithm," *Discrete Appl. Math.*, vol. 261, pp. 28–39, May 2019.

[15] A. Montero, I. Méndez-Díaz, and J. J. Miranda-Bront, "An integer programming approach for the time-dependent traveling salesman problem with time Windows," *Comput. Oper. Res.*, vol. 88, pp. 280–289, Dec. 2017.

[16] D. M. Vu, M. Hewitt, N. Boland, and M. Savelsbergh, "Dynamic discretization discovery for solving the time-dependent traveling salesman problem with time Windows," *Transp. Sci.*, vol. 54, no. 3, pp. 703–720, May 2020.

[17] C. Pralet, "Iterated maximum large neighborhood search for the traveling salesman problem with time windows and its time-dependent version," *Comput. Oper. Res.*, vol. 150, Feb. 2023, Art. no. 106078.

[18] C. S. Helvig, G. Robins, and A. Zelikovsky, "The moving-target traveling salesman problem," *J. Algorithms*, vol. 49, no. 1, pp. 153–174, Oct. 2003.

[19] R. Montemanni, J. Barta, M. Mastrolilli, and L. M. Gambardella, "The robust traveling salesman problem with interval data," *Transp. Sci.*, vol. 41, no. 3, pp. 366–381, Aug. 2007.

[20] Y. Bengio, A. Lodi, and A. Prouvost, "Machine learning for combinatorial optimization: A methodological tour d'horizon," *Eur. J. Oper. Res.*, vol. 290, no. 2, pp. 405–421, 2020.

[21] O. Vinyals, M. Fortunato, and N. Jaitly, "Pointer networks," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 28, 2015, pp. 1–9.

[22] I. Bello, H. Pham, Q. V. Le, M. Norouzi, and S. Bengio, "Neural combinatorial optimization with reinforcement learning," 2016, *arXiv:1611.09940*.

[23] T. Adamo, T. Calogiuri, G. Ghiani, A. Grieco, E. Guerriero, and E. Manni, "Neighborhood synthesis from an ensemble of MIP and CP models," in *Proc. Int. Conf. Learn. Intell. Optim.* Cham, Switzerland: Springer, 2016, pp. 221–226.

[24] V. Uslan and I. O. Bucak, "A comparative study of machine learning heuristic algorithms to solve the traveling salesman problem," in *Proc. 3rd Int. Conf. Appl. Digit. Inf. Web Technol. (ICADIWT)*, 2010, pp. 12–14.

[25] G. Ghiani, T. Adamo, P. Greco, and E. Guerriero, ''Lifting the performance of a heuristic for the time-dependent travelling salesman problem through machine learning. *Algorithms*, vol. 13, no. 12, p. 340, 2020.
[26] T. Adamo, G. Ghiani, and E. Guerriero, ''On path ranking in time-dependent graphs,'' *Comput. Oper. Res.*, vol. 135, Nov. 2021, Art. no. 105446.
[27] S. Ichoua, M. Gendreau, and J.-Y. Potvin, ''Vehicle dispatching with time-dependent travel times,'' *Eur. J. Oper. Res.*, vol. 144, no. 2, pp. 379–396, Jan. 2003.
[28] C. C. Aggarwal, *Neural Networks and Deep Learning*. Berlin, Germany: Springer, 2018.
[29] J. MacQueen, ''Some methods for classification and analysis of multivariate observations,'' in *Proc. 5th Berkeley Symp. Math. Statist. Prob.*, vol. 1, Oakland, CA, USA, Jun. 1967, pp. 281–297.
[30] G. Carpaneto, M. Dell'Amico, and P. Toth, ''Exact solution of large-scale, asymmetric traveling salesman problems,'' *ACM Trans. Math. Softw.*, vol. 21, no. 4, pp. 394–409, Dec. 1995.
[31] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. Cambridge, MA, USA: MIT Press, 2016.

TOMMASO ADAMO received the Graduate degree in computer engineering and the Ph.D. degree in engineering of complex systems from the University of Salento, in 2013 and 2017, respectively. He is currently a Researcher in operations research with the University of Salento. His research interests include combinatorial optimization models and machine learning in vehicle routing problems.

GIANPAOLO GHIANI is currently a Professor in operations research at the University of Salento, Lecce, Italy. He was a Postdoctoral Researcher at the Groupe d'Etudes et de Recherche en Analyse des Decisions (GERAD), Universitede Montreal, Montreal, Canada, from 1998 to 1999. He has published over 70 articles in a variety of international journals. His doctoral thesis was awarded the Transportation Science Dissertation Award from INFORMS, in 1998. He has coauthored the textbook *Introduction to Logistics Systems Management* (Wiley, 2022). His research interests include the field of decision support systems (including methodologies at the boundary between operations research and artificial intelligence) with an emphasis on applications to logistic systems planning and control.

PIERPAOLO GRECO received the B.S. degree in civil engineering and the Ph.D. degree in complex system engineering from the University of Salento, in 2018 and 2022, respectively. His research interests include operations research in vehicle routing problem and heuristic algorithms.

EMANUELA GUERRIERO received the Laurea (M.Sc.) degree in computer engineering from the University of Lecce, in 1999, and the Ph.D. degree in operations research from the University of Calabria, in 2003. She is currently an Associate Professor at the Department of Engineering for Innovation, University of Salento. Her research interests include combinatorial optimization for planning and scheduling problems, with applications in manufacturing and logistics, problem-solving methods lying at the boundary between operations research and artificial intelligence. She has published more than 37 articles in international journals, including, *International Transactions in Operational Research*, *European Journal of Operational Research*, *Parallel Computing*, *International Journal of Production Research*, *International Journal of Production Economics*, *Discrete Applied Mathematics*, *Transportation Science*, *4OR*, and *Computers and Operations Research*.