algorithms

MDPI

Article

Lifting the Performance of a Heuristic for the Time-Dependent Travelling Salesman Problem through Machine Learning

Gianpaolo Ghiani *, Tommaso Adamo, Pierpaolo Greco and Emanuela Guerriero

Dipartimento di Ingegneria dell'Innovazione, Università del Salento, via per Monteroni, 73100 Lecce, Italy; tommaso.adamo@unisalento.it (T.A.); pierpaolo.greco@unisalento.it (P.G.); emanuela.guerriero@unisalento.it (E.G.)

* Correspondence: gianpaolo.ghiani@unisalento.it

Received: 21 October 2020 ; Accepted: 15 November 2020; Published: 14 December 2020

Abstract: In recent years, there have been several attempts to use machine learning techniques to improve the performance of exact and approximate optimization algorithms. Along this line of research, the present paper shows how supervised and unsupervised techniques can be used to improve the quality of the solutions generated by a heuristic for the Time-Dependent Travelling Salesman Problem with no increased computing time. This can be useful in a real-time setting where a speed update (or the arrival of a new customer request) may lead to the reoptimization of the planned route. The main contribution of this work is to show how to reuse the information gained in those settings in which instances with similar features have to be solved over and over again, as it is customary in distribution management. We use a method based on the nearest neighbor procedure (supervised learning) and the K-means algorithm with the Euclidean distance (unsupervised learning). In order to show the effectiveness of this approach, the computational experiments have been carried out for the dataset generated based on the real travel time functions of two European cities: Paris and London. The overall average improvement of our heuristic over the classical nearest neighbor procedure is about 5% for London, and about 4% for Paris.

Keywords: machine learning; travelling salesman problem; time-dependent travel times

1. Introduction

The development of efficient optimization methods for real-life problems is a rich and interesting research area. If the problem is enough easy with very effective bounds and a limited dimensionality, the analyst will be able to write a formal mathematical model and solve it with a general-purpose black box solver. This process usually takes few days of work. Otherwise, the analyst have to design and develop an ad-hoc heuristic that requires a huge effort during the tuning and testing phase. This time the development process requires weeks or months depending on the experience of the analyst in the sector and on the problem size. The use of approximate methods to solve combinatorial optimization problems has received an increasing attention in the last years. Approximate methods sacrifice the optimality guarantee for the sake of getting good solutions in a significantly reduced amount of time. Constructive algorithms are a simple example of approximate methods, which generate solutions from scratch by adding components to an initially empty partial solution, until a feasible solution is complete. Nowadays, a class of approximate algorithms, commonly called metaheuristics, has emerged. In short, we could say that metaheuristics are high level strategies for exploring search spaces by using different methods to efficiently produce high-quality solutions. Techniques which constitute metaheuristic algorithms range from simple local search procedures to complex learning processes. They are

non-deterministic, not problem-specific, they can use domain-specific heuristics that are controlled by the upper level strategy, they commonly incorporate mechanisms to escape from local optima and finally they can take advantage of the search experience (adding some form of memory) to guide the search. A broad coverage of the concepts, implementations, and applications in metaheuristics can be found in Gendreau et al. [1]. In Adamo et al. [2] was proposed a framework to automatically design efficient neighborhood structures for any metaheuristic from a formal mathematical model and a reference instance population. There is a wide variety of metaheuristics. Basically we can distinguish between single solution approaches, which focus on modify and improve a single candidate solution (simulated annealing, iterated local search, variable neighborhood search, and guided local search), and population-based approaches, that maintain and improve multiple candidate solutions, often using population characteristics to guide the search (evolutionary computation, genetic algorithms, and particle swarm optimization). Evolutionary algorithms, particle swarm optimization, differential evolution, ant colony optimization and their variants dominate the field of nature-inspired metaheuristics. Swarm intelligence has been proved as a technique, based on observations of nature, which can solve NP-hard computational problems. Some concept and metaheuristics (the particle swarm optimization algorithm and the ant colony optimization method) belonging to the smart intelligence are presented in Slowik and Kwasnicka [3]. It is gaining popularity in solving different optimization problems and has been used successfully for feature selection in some applications. A comprehensive literature review of swarm intelligence algorithms and a detailed definition of taxonomic categories was reported in [4]. Swarm optimization have been also successfully applied to Social Cognitive Radio Network by Anandakumar and Umamaheswari [5], which proposed a technique that utilizes machine learning methods to adapt the environmental changes, create its own knowledge base and adjust its functionality for making dynamic data and network handover decisions. Zhao et al. [6] developed a wind energy decision system. Evolutionary algorithms are population-based metaheuristics that uses mechanisms inspired by biological evolution, such as reproduction, mutation, recombination, and selection. A self-adaptive Evolutionary Algorithm is proposed in Dulebenets et al. [7] for the berth scheduling problem. Pasha et al. [8] provided a full comparison among Variable Neighborhood Search, Tabu Search, Simulated Annealing and Evolutionary Algorithm to solve a model for large-scale problem instances of the Vehicle Routing Problem. They demonstrated that the Evolutionary Algorithm outperforms the other metaheuristic algorithms developed for the model.

In recent years there has been a flourishing of articles that try leveraging Machine Learning (ML) to solve combinatorial optimization problems. In this context, ML may help substitute heavy computations by a fast approximation. This is the case of learning variable and node selection in branch-based algorithms for Mixed-Integer Linear Programming (MIP). See Lodi and Zarpellon [9] for a recent review. A wider perspective is taken by Bengio et al. [10] that identify three broad approaches to leveraging machine learning for combinatorial optimization problems: learning alongside optimization algorithms, learning to configure optimization algorithms, and end-to-end learning to approximately solve optimization problems. Some results of this line of research have been recently used in industry: in November 2019, following the work of Bonami et al. [11], IBM announced that version 12.10 of its well-known commercial optimization solver CPLEX implements, for the first time, a ML-based classifier to make automatic decisions over some algorithmic settings. In particular, an ML-based classifier is invoked by default to decide if the binary component of a Mixed-Integer Quadratic Optimization problem should benefit from the application of a linearization procedure, which transforms the problem into a MIP problem.

In this paper, we make use of supervised and unsupervised techniques to improve the quality of the solutions generated by a heuristic for the Time-Dependent Travelling Salesman Problem with no increased computing time. This can be useful in a real-time setting where a speed update (or the arrival of a new customer request) may lead to the reoptimization of the planned route. As far as we know, this is the first attempt to use ML to solve a time-dependent routing problem. For a comparative

analysis of machine learning heuristics for solving the classical (time-invariant) Travelling Salesman Problem, see Uslan and Bucak [12].

In Time-Dependent Vehicle Routing Problems, the aim is to design routes for a fleet of vehicles on a graph whose arc traversal times vary over time, in order to optimize a given criterion, possibly subject to side constraints. See Gendreau et al. [13] for a review of the field.

In this paper, we study the Time-Dependent Travelling Salesman Problem (TDTSP) which amounts to find a Hamiltonian tour of least total duration on a given time-dependent graph. The TDTSP was first addressed by Malandraki and Daskin [14]; they proposed a *Mixed Integer Programming* (MIP) formulation for the problem. Subsequently, Malandraki and Dial [15] developed an approximate dynamic programming algorithm, whilst Li et al. [16] presented two heuristics. Schneider [17] and Harwood et al. [18] presented some meta-heuristic approaches to solve the TDTSP. Cordeau et al. [19] showed some properties of the TDTSP and derived lower and upper bounding procedures; they also devised a set of valid inequalities used in a branch-and-cut algorithm. Arigliano et al. [20] derived some properties of the problem that were used in a branch-and-bound algorithm that outperformed the Cordeau et al. [19] branch-and-cut procedure. In Adamo et al. [21] a parameterized family of lower bounds was developed to enhance this branch-and-bound approach. Moreover, Melgarejo et al. [22] presented a new global constraint useful in a *Constraint Programming* approach.

Variants of the TDTSP have been studied by [23–26] (*TDTSP with Time Windows*), by [27] (*Moving-Target TSP*), by [28] (*Robust TSP with Interval Data*), and by [29–35] (*Single Machine Time-Dependent Scheduling Problem*).

This paper is organized as follows. In Section 2 we present some properties and some background information on the study area. In Section 3 we describe our ML approach. In Section 4 we describe computational experiments on the graphs of two European cities (London and Paris). Finally, we draw some conclusions in Section 5.

2. State of the Art

Let $G = (V, A)$ be a directed graph, where $V = \{1, 2, \ldots, n\}$ is a set of vertices and $A \subseteq V \times V$ a set of arcs. With each arch $(i, j) \in A$ is associated a travel time function $\tau_{ij}(t)$ representing the time that a vehicle, departing from i at time instant t, takes to traverse the arc and finally arrive at vertex j. Let $\mathcal{T} = [0, T]$ be the time horizon (e.g., an 8-h period of a typical working day). Given a start time t and path composed by a sequence of k nodes, i.e., $p_n = (i_0, i_1, \ldots, i_n)$ with $i_k \in V$ and $k = 0, \ldots, n$, the time needed to travel from node i_0 to node i_n can be computed recursively as:

$$z(p_k, t) = z(p_{k-1}, t) + \tau_{i_{k-1}, i_k}(t + z(p_{k-1}, t) \quad k = 1, \ldots, n$$

and initialization $z(p_0, t) = 0$. Without loss of generality, we assume that the travel time functions are piecewise linear and satisfy the first-in-first-out (FIFO) property, i.e., the arrival time is a strictly monotonic function of the starting time. Ghiani and Guerriero [36] proved that any continuous piecewise linear travel time function $\tau_{ij}(t)$, satisfying the FIFO property, can be generated from the IGP model. In the IGP model the speed of the vehicle is not constant over the entire length of arc (i, j) but it changes when the boundaries between two consecutive time periods is crossed. For any arc the time horizon $\mathcal{T} = [0, T]$ is divided into H_{ij} time slots $[T_{ijh}, T_{ij(h+1)}]$, $h = 0, 1, \ldots, H_{ij} - 1$. Let v_{ijh} be the travel speed between node i and node j in time slot h and L_{ij} be the distance between the two vertices i and j. They decomposed the travel speed for an arc (i, j) as:

$$v_{ijh} = \delta_{ijh} b_h u_{ij}, \tag{1}$$

where:

- $u_{ij} \geq 0$ is the maximum travel speed across arc $(i, j) \in A$ in all times.
- $0 < b_h \leq 1$ is the best congestion factor during interval h ($[T_h, T_{h+1}]$);
- $0 < \delta_{ijh} \leq 1$ is the degradation of the congestion factor of arc (i, j) in time slot h.

The Time-Dependent Travelling Salesman Problem amounts to find a Hamiltonian tour of least total duration on a given time-dependent graph. Cordeau et al. [19] proposed a lower bound for the TDTSP based on the IGP model. In particular, they proved that when $\Delta = \min_{i,j,h} \delta_{ijh} = 1$ solving the time-dependent problem is equivalent to solve its time-invariant counterpart. In [36], the authors pointed out that in IGP model the relationship among travel speeds $v_{ij}(t)$, times $\tau_{ij}(t)$ and lengths L_{ij} can be expressed as follows:

$$L_{ij} = \int_{t}^{t+\tau_{ij}(t)} v_{ij}(\mu) d\mu, \tag{2}$$

where it is worth noting that the IGP parameters, L_{ij} and $v_{ij}(t)$, can be multiplied by any positive common factor without changing the travel time functions $\tau_{ij}(t)$. This implies that L_{ij} can be any positive number and, consequently, the speed factorization (1) is not uniquely defined. Indeed, Ref. [21] proved that there exists a different speed factorization for each value of u_{ij} such that $u_{ij} \geq \max_{h} v_{ijh}$. Adamo et al. [37] generalize all these considerations in order to improve the previous lower bounds.

3. Leveraging Machine Learning in a TDTSP Heuristic

In Section 1 we have recalled the reasons around the growing interest in applying Machine Learning models to optimization problems. The aim of this section is show how to embed some information gained through a ML algorithm into a simple constructive heuristic. The goal is to improve its performance in those settings in which instances with similar features have to be solved over and over again, as it is customary in distribution management. Instead of starting every time from scratch, we want to insert a learning mechanism inside the heuristic in such a way it can benefit from previous runs on other instances with similar features.

The constructive heuristic we are going to use as a baseline is the well-known *nearest neighbor procedure* (NNP) that is suitable to be used in a real-time setting due to its speed. Algorithm 1 reports its pseudocode, where V_1 is the set of already visited customers.

Algorithm 1 Baseline heuristic.

1: **function** NN(V)
2: $i \leftarrow 0$
3: $t \leftarrow 0$
4: $V_1 \leftarrow \{0\}$
5: **while** $|V_1| \leq |V|$ **do**
6: $j^* \leftarrow arg \min_{j \in V \setminus V_1} \tau_{ij}(t)$
7: $V_1 \leftarrow V_1 \bigcup \{j^*\}$
8: $t \leftarrow t + \tau_{ij^*}(t)$
9: $i \leftarrow j^*$
10: **end while**
11: $t \leftarrow t + \tau_{i0}(t)$
12: **return** t, V_1

In order to take advantage of the predictive capabilities of supervised ML techniques, the nearest neighbor heuristic has been modified by the introduction of two parameters: α and f_j. Parameter f_j is a prediction (obtained through a supervised ML method) of the *expected time of arrival* (ETA) at customer j in an optimal solution, and $\alpha \in [0, 1]$ is a convex combination coefficient. At each iteration, a node is selected using a modification of instruction 6 in Figure 1 as follows:

$$j^* \leftarrow arg \min_{j \in V \setminus V_1} \alpha \cdot [t + \tau_{ij}(t)] + (1 - \alpha) \cdot |f_j - t - \tau_{ij}(t)| \tag{3}$$

Hence the choice of the next customer to be visited is guided by a convex combination of two terms: the arrival time $t + \tau_{ij}(t)$ at the next customer (as it is typical of the NNP) and an error measure

$|f_j - t - \tau_{ij}(t)|$ of forecast f_j, if j is chosen as the next customer. Parameter $\alpha \in [0,1]$ is used to balance the cost function $t + \tau_{ijh}$ with the correction factor depending on f_j. In few words, if α is zero the heuristic takes its own decisions only based on the correction factor; on the other hand, if α is 1 the heuristic is the well-known NNP, no matter how f_j looks like. In the latter case, the heuristic takes no advantage from previous runs. Hence, α can be seen as a *learning factor*.

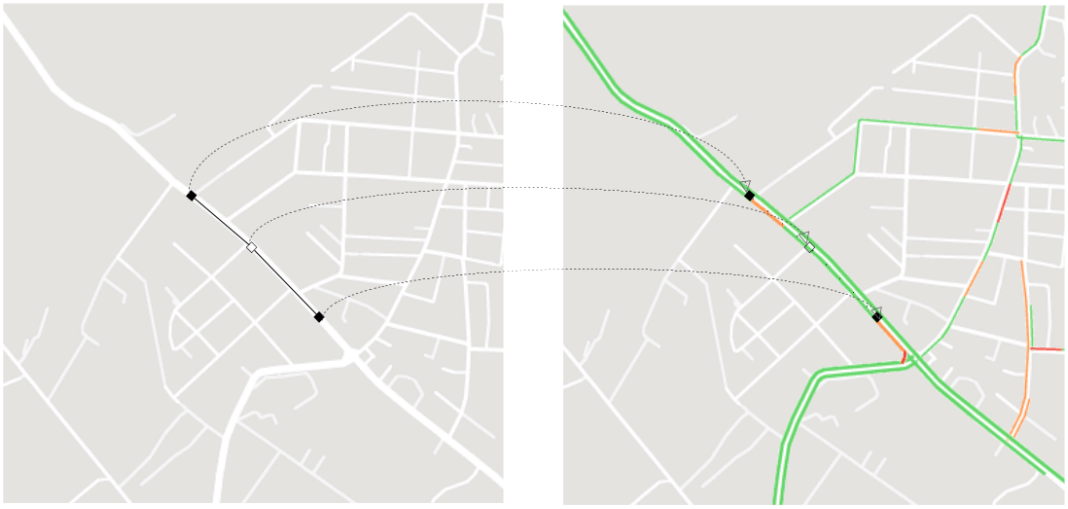

Figure 1. Extracting travel times from a snapshot.

3.1. ETA Estimation

In order to estimate the ETA of a customer j in an optimal solution, an artificial neural network (ANN) is used in conjunction with an exact algorithm for the TDTSP [21]. The ANN we chose is a *Multilayer Perceptron Regressor* (MPR) [38]. An MPR consists of at least three layers of nodes: an input layer, one or more hidden layer and an output layer. Except for the input nodes, each node uses a nonlinear activation function.

For each instance of the training set, the exact algorithm is used to obtain the arrival times at the customers. In order to overcome the drawbacks described in the following subsection, the service territory is partitioned in K zones and, for each training instance, an average zone ETA, dubbed $ZETA_k$, is computed for each zone $k = 1, \ldots, K$.

The neural network has k inputs and k outputs: the inputs are constituted by the customer distribution in the network (the number n_k of customers in each of the k zones); the outputs are the k $ZETA_k$ estimates ($k = 1, \ldots, K$).

3.2. Customer Aggregation

In order to make the ETA predictions accurate, customers must be aggregated and the service territory divided into a number of zones K. Of course, if K is large the predictions are expected to be more accurate but the training phase would require a huge number of instances. On the other hand, if K is small, the natural variability of the ETA inside a zone is large which has a detrimental effect on the ETA estimation of individual customers.

The customer aggregation is an unsupervised learning technique that amo-unts to partition the customers of the training set into K clusters of equal variance, minimizing the sum of intra-cluster Euclidean distances. In our experimentation, we used a K-means algorithm [39]. Since the geographic coordinate system is associated with the geodesic surface of the Earth, we projected customers locations to a new planar reference system according to the EPSG projection recommended for the zone at hand. Then the projected customers have been clustered using the K-means algorithm. The optimal number of zones K was determined in a preliminary experimentation.

4. Computational Experiments

Our computational experiments have been aimed to evaluate whether the use of predictions made by ML techniques can be beneficial for a routing heuristic in a time-dependent context. To this purpose we have implemented both the baseline heuristic (NNP) and the ML-enhanced heuristic (referred to as ML-NNP in the following). Python (version 3.6) was used for both. The Multilayer Perceptron Regressor implementation was taken from the *sklearn* neural network library (method MLPRegressor) while the k-means implementation came from the *sklearn* cluster library (k-means method). The training instances were solved to optimality (or near-optimality) using a Java implementation of the branch-and-bound scheme proposed in [19]. A time limit of an hour was imposed. All the codes have been tested on a Linux machine clocked at 2.67 GHz and equipped with 8 GB of RAM. We first describe the instances we have used in our experimentations.

4.1. Instances

Although Time-Dependent Vehicle Routing problems have received increased attention from the scientific community in recent years, there is still a lack of realistic instances. To overcome this aspect, we have generated a new set of instances based on the real travel time functions of two major European cities: Paris and London.

During the last decade, the popularity of mobile technology and location services allowed the creation of high quality large real-time and historical floating vehicle datasets. These data are continuously collected by millions of users that anonymously send their GPS positions to services like Google Traffic or TomTom Live. Only these IT big players have the availability of time-dependent data. As a result, there is no complete dataset freely available to the entire research community.

We extracted (approximate) travel time functions as follows. Given a road map from OpenStreetMap (in XML or PBF format) we first extracted the topology of graph G in terms of: vertices and their geographical coordinates (latitude and longitude), arcs with attributes like length, maximum speed u_{ij} (derived from tags or default street type speed), geometry (pillar and tower nodes). The API provided from the GraphHopper project has been used to accomplish tasks as OSM importation, road graph generation and routing between customers. We extended this time-invariant graph representation by assigning a traffic profile only to time-dependent arcs. In particular, traffic data were obtained through a two step procedure. Firstly, given the geographic coordinates of two points at a specific zoom level, we defined a bounding box around the reference Earth area. Therefore, at the start of every time slot h we captured a snapshot of the current map viewport augmented with the real-time traffic layer. Next, for each arc (i, j) belonging in the bounding box we subdivided its geometry in uniform length segments. We denoted the set of all segments endpoints for arc (i, j) by S_{ij}. For each point $s \in S_{ij}$ we projected the geographic coordinates over a pixel s' of the image plane (Figure 1). We used the Haversine formula to measure Earth distances. For each period h the pixel s' can assume a different color in the corresponding h-th snapshot. The color is classified using the CIEDE2000 color distance [40] from some reference color with a preassigned traffic jam factor as showed in Table 1.

Table 1. Color map used to extract travel times.

Ref. Color	Color Name	Jam_1	Jam_2	Description
#84CA50	green	1.0	1.0	high speed
#F07D02	orange	0.7	0.3	medium speed
#E60000	red	0.5	0.2	low speed
#9E1313	brown	0.3	0.1	very low speed
#EBE8DE	gray	–	–	background
#FFFFFF	white	1.0	1.0	freeflow

If s' did not belong to a street, we started a neighborhood exploration procedure in order to identify the nearest valid color. This task is executed to repair small alignment errors between the osm map and the traffic layer image. After classification, the traffic jam factor of arc (i, j) was computed as the mean of the jam factors associated to each geographical point $s \in S_{ij}$. An arc was deemed to be time-dependent if it had at least one traffic jam factor strictly lower than 1 in the time horizon.

4.2. Parameter Tuning

We have performed a preliminary tuning with the aim to select the most appropriate combination of parameters:

- the activation function;
- the number of hidden layers;
- the training algorithm;
- the learning rate;
- the maximum number of iterations in the training phase;
- the number of zones.

Table 2 summarizes the investigated values for each parameter.

Table 2. Investigated parameter values.

Parameter	Possible Values
Activation Function	tanh, logistic, relu
Number of Hidden Layers	(3) (4) (5) (10) (15) (20) (30) (40) (3,3) (4,4) (5,5) (10,10) (15,15) (20,20) (30,30) (40,40)
Solver	sgd, lbfgs, adam
Learning Rate	constant, inverse scaling, adaptive
Max. iterations	200, 300, 400, 500
Number of zones	5, 6, 7, 8, 9, 10, 11, 12

Our datasets contained approximately 6–700 instances with 50 customers each: 90% has been assigned to the training set, while the remaining 10% to the test set. The best results, in terms of strength of captured relationships, were obtained by the following neural network settings: three layers, hyperbolic tangent activation function, five neurons in the hidden layer, LBFGS solver and constant learning rate.

As far as customer aggregation is concerned, it was necessary to project customers locations to a new planar reference system. According to the EPSG projection, we used the Transverse Mercator Projection for London and the Lambert Zone II Projection for Paris.

For London, 8 clusters gave the best results in terms of coefficient of determination (R^2), whilst for Paris 6 zones were the best case for neural network performance. Tables 3 and 4 summarize the neural network mean errors (in minutes) for each zone. The R^2 scores (=0.53 for the London instances and =0.60 for the Paris instances) suggest a moderate effect size.

Table 3. Mean errors in 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 4. Mean errors in the 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

4.3. Computational Results

The computational results are presented in Tables 5 and 6. For each of the two testsets, we report:

- the name of the test instance,
- the objective value z_0 in minutes of the NNP solution,
- the objective value z_1 in minutes of the ML-NNP solution,
- the percentage of improvement DEV of z_1 with respect to z_0.

Table 5. Computational results for the London testset.

Instance	z_0	z_1	DEV%	Instance	z_0	z_1	DEV%
10_I_1	406.13	406.13	0.00	10_I_10	484.56	447.78	7.59
10_I_11	449.64	449.64	0.00	10_I_12	614.67	524.43	14.68
10_I_13	444.23	443.87	0.08	10_I_14	486.83	476.72	2.08
10_I_15	453.19	451.05	0.47	10_I_16	515.56	469.16	9.00
10_I_17	517.51	497.65	3.84	10_I_19	466.79	466.79	0.00
10_I_2	526.66	447.68	15.00	10_I_20	428.63	414.95	3.19
10_I_23	491.50	447.15	9.02	10_I_24	510.02	473.05	7.25
10_I_25	470.01	468.86	0.24	10_I_26	489.47	484.20	1.08
10_I_27	492.53	458.75	6.86	10_I_28	507.26	456.07	10.09
10_I_29	440.37	436.33	0.92	10_I_30	449.90	449.90	0.00
10_I_31	474.33	443.42	6.52	10_I_32	464.37	443.46	4.50
10_I_33	401.02	390.20	2.70	10_I_34	422.61	401.68	4.95
10_I_36	502.89	441.57	12.19	10_I_37	464.31	460.60	0.80
10_I_38	540.78	527.42	2.47	10_I_39	531.83	489.67	7.93
10_I_40	503.19	503.06	0.03	10_I_41	468.83	460.46	1.78
10_I_5	480.03	439.48	8.45	10_I_6	457.51	448.56	1.96
10_I_7	429.42	429.42	0.00	10_I_9	420.89	420.89	0.00
1_I_2	497.99	479.33	3.75	1_I_26	472.87	436.01	7.80
1_I_27	491.03	442.35	9.91	1_I_28	448.76	440.16	1.92
1_I_29	476.59	428.66	10.06	1_I_3	473.71	434.48	8.28
1_I_30	415.91	413.30	0.63	1_I_31	477.35	472.73	0.97
1_I_32	503.70	457.25	9.22	1_I_33	423.53	396.61	6.36
1_I_34	498.12	473.91	4.86	1_I_35	411.64	411.64	0.00
1_I_36	497.65	463.28	6.91	1_I_37	447.20	440.50	1.50
1_I_39	468.47	458.62	2.10	1_I_4	460.05	458.39	0.36
1_I_40	468.46	457.96	2.24	1_I_42	469.45	452.29	3.66
1_I_44	466.47	411.89	11.70	1_I_45	485.20	480.50	0.97
1_I_46	473.24	440.80	6.85	1_I_47	462.35	451.22	2.41
1_I_48	493.13	431.62	12.47	1_I_49	481.59	453.77	5.78
1_I_5	427.22	403.86	5.47	1_I_50	487.52	442.95	9.14
1_I_51	449.49	429.09	4.54	1_I_53	404.83	363.48	10.21

In our implementations, the maximum running time of both algorithms is only a few seconds or even a fraction of a second. Therefore we prefer not to report these figures. As shown in Tables 5 and 6,

the overall average improvement of ML-NNP over NNP is 4.77% for London, while 3.87% for Paris. It is worth noting that in the worst case ML-NNP gives the same performance of NNP, while in the best case can produce a solution with an improvement up to 15% (more than 70 min over a typical working day of 8 h).

Table 6. Computational results for the Paris testset.

Instance	z_0	z_1	*DEV*%	Instance	z_0	z_1	*DEV*%
0_I_0	328.71	324.94	1.15	0_I_100	318.16	317.09	0.34
0_I_101	309.36	305.77	1.16	0_I_102	317.81	295.56	7.00
0_I_103	349.52	345.77	1.07	0_I_104	363.87	330.92	9.06
0_I_105	366.60	356.27	2.82	0_I_106	344.89	321.21	6.87
0_I_107	349.68	330.46	5.50	0_I_108	322.95	308.71	4.41
0_I_109	355.49	317.67	10.64	0_I_10	341.40	325.10	4.77
0_I_110	302.99	302.99	0.00	0_I_111	344.07	336.45	2.21
0_I_112	361.19	329.37	8.81	0_I_113	339.78	328.51	3.32
0_I_114	326.10	318.03	2.48	0_I_115	328.84	327.61	0.37
0_I_116	337.34	315.94	6.34	0_I_117	334.44	331.77	0.80
0_I_118	329.44	307.59	6.63	0_I_119	349.89	341.47	2.41
0_I_11	353.87	347.72	1.74	0_I_120	340.53	332.81	2.27
0_I_121	336.74	317.04	5.85	0_I_122	316.01	292.70	7.38
0_I_123	379.26	336.27	11.33	0_I_124	344.95	328.87	4.66
0_I_125	314.78	314.78	0.00	0_I_126	349.34	329.46	5.69
0_I_127	337.81	337.81	0.00	0_I_128	338.07	317.47	6.09
0_I_129	324.69	319.44	1.62	0_I_12	323.21	323.21	0.00
0_I_130	357.66	336.79	5.83	0_I_131	305.09	305.09	0.00
0_I_132	299.75	279.76	6.67	0_I_133	325.39	316.83	2.63
0_I_134	348.87	312.39	10.46	0_I_135	376.12	334.26	11.13
0_I_136	320.55	311.08	2.96	0_I_137	320.33	314.02	1.97
0_I_138	320.04	308.52	3.60	0_I_139	335.29	332.52	0.83
0_I_13	329.99	301.86	8.52	0_I_140	317.49	317.49	0.00
0_I_141	364.98	336.72	7.74	0_I_142	323.90	319.11	1.48
0_I_143	302.36	302.15	0.07	0_I_144	310.60	310.60	0.00
0_I_145	276.94	267.63	3.36	0_I_146	344.77	303.90	11.85
0_I_147	360.36	328.85	8.75	0_I_148	296.80	292.53	1.44
0_I_149	357.70	342.24	4.32	0_I_14	307.80	307.80	0.00
0_I_150	333.85	324.11	2.92	0_I_151	349.22	319.25	8.58
0_I_152	333.04	328.54	1.35	0_I_153	292.33	291.41	0.32
0_I_154	347.20	320.46	7.70	0_I_155	383.86	363.29	5.36
0_I_156	322.72	312.77	3.08	0_I_157	317.25	312.05	1.64
0_I_159	324.43	321.24	0.98	0_I_15	344.14	329.78	4.17
0_I_160	302.85	302.85	0.00	0_I_161	345.81	344.50	0.38
0_I_162	370.24	361.01	2.49	0_I_163	317.24	313.21	1.27
0_I_164	300.10	300.10	0.00	0_I_165	320.44	314.03	2.00
0_I_166	366.52	340.85	7.00	0_I_168	333.84	327.67	1.85
0_I_169	326.60	316.67	3.04	0_I_16	376.84	325.67	13.58
0_I_17	334.56	327.77	2.03	0_I_1	313.16	300.76	3.96

5. Conclusions

In this paper, we have leveraged a mix of unsupervised and supervised Machine Learning techniques in a heuristic for the Time-Dependent Travelling Salesman Problem. Our approach makes use of ETA predictions provided by a feedforward neural network trained on past instances solved to optimality or near-optimality. Computational results on two European cities show the advantage of embedding ML methods into an optimization algorithm. As far as we know, this is the first attempt to use ML to tackle a Time-Dependent Vehicle Routing problem. The main limitations of such approach are related to the definition of a population of training instances and the determination of the best solution for that instances; the best known solution algorithm could need several hours to determine such a solution.

Future research could be focused on the definition of new features for the neural network in order to provide a more reasonable and substantial effect in the heuristic; moreover, other machine learning strategies could be tried (for instance decision trees). Finally, a new fast heuristic, probably better than nearest neighbor algorithm, could be produced by embedding the ETA estimation in the linear programming model introduced by Adamo et al. [37].

Author Contributions: Methodology, P.G.; software, E.G.; formal analysis, T.A.; project administration, G.G. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Acknowledgments: This research was partially supported by the Ministero dell'Istruzione, dell'Università e della Ricerca Scientifica (MIUR) of Italy. This support is gratefully acknowledged. The authors also thank Federico Liquori and Marco D'Amato for their help with programming.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Michel, G.; Potvin, J.Y. (Eds.) *Handbook of Metaheuristics*; Springer: Berlin/Heisenberg, Germany, 2010; Volume 2.
2. Adamo, T.; Ghiani, G.; Grieco, A.; Guerriero, E.; Manni, E. MIP neighborhood synthesis through semantic feature extraction and automatic algorithm configuration. *Comput. Oper. Res.* **2017**, *83*, 106–119.
3. Slowik, A.; Kwasnicka, H. Nature inspired methods and their industry applications—Swarm intelligence algorithms. *IEEE Trans. Ind. Inform.* **2018**, *14*, 1004–1015. [CrossRef]
4. Brezočnik, L.; Fister, I.; Podgorelec, V. Swarm intelligence algorithms for feature selection: A review. *Appl. Sci.* **2018**, *8*, 1521. [CrossRef]
5. An akumar, H.; Umamaheswari, K. A bio-inspired swarm intelligence technique for social aware cognitive radio handovers. *Comput. Electr. Eng.* **2018**, *71*, 925–937. [CrossRef]
6. Zhao, X.; Wang, C.; Su, J.; Wang, J. Research and application based on the swarm intelligence algorithm and artificial intelligence for wind farm decision system. *Renew. Energy* **2019**, *134*, 681–697. [CrossRef]
7. Dulebenets, M.A.; Kavoosi, M.; Abioye, O.; Pasha, J. A self-adaptive evolutionary algorithm for the berth scheduling problem: Towards efficient parameter control. *Algorithms* **2018**, *11*, 100. [CrossRef]
8. Pasha, J.; Dulebenets, M.A.; Kavoosi, M.; Abioye, O.F.; Wang, H.; Guo, W. An Optimization Model and Solution Algorithms for the Vehicle Routing Problem With a "Factory-in-a-Box". *IEEE Access* **2020**, *8*, 134743–134763. [CrossRef]
9. Lodi, A.; Zarpellon, G. On learning and branching: A survey. *Top* **2017**, *25*, 207–236. [CrossRef]
10. Bengio, Y.; Lodi, A.; Prouvost, A. Machine learning for combinatorial optimization: A methodological tour d'horizon. *arXiv* **2018**, arXiv:abs/1811.06128.
11. Philpott, A.B. Continuous-time shortest path problems and linear programming. *SIAM J. Control. Optim.* **1994**, *32*, 538–552. [CrossRef]
12. Uslan, V.; Bucak, I.O. A comparative study of machine learning heuristic algorithms to solve the traveling salesman problem. In Proceedings of the 3rd International Conference on the Applications of Digital Information Web Technologies (ICADIWT), Istanbul, Turkey, 12–14 July 2010.

13. Gendreau, M.; Ghiani, G.; Guerriero, E. Time-dependent routing problems: A review. *Comput. Oper. Res.* **2015**, *64*, 189–197. [CrossRef]
14. Malandraki, C.; Daskin, M.S. Time Dependent Vehicle Routing Problems: Formulations, Properties and Heuristic Algorithms. *Transp. Ence* **1992**, *26*, 185–200. [CrossRef]
15. Malandraki, C.; Dial, R.B. A restricted dynamic programming heuristic algorithm for the time dependent traveling salesman problem. *Eur. J. Oper. Res.* **1996**, *90*, 45–55. [CrossRef]
16. Li, F.; Golden, B.; Wasil, E. Solving the time dependent traveling salesman problem. In *The Next Wave in Computing, Optimization, and Decision Technologies*; Volume 29 of Operations Research/Computer Science Interfaces Series; Sharda, R., Voß, S., Golden, B., Raghavan, S., Wasil, E., Eds.; Springer: Berlin/Heisenberg, Germany, 2005; pp. 163–182.
17. Schneider, J. The time-dependent traveling salesman problem. *Phys. Stat. Mech. Appl.* **2002**, *314*, 151–155. [CrossRef]
18. Harwood, K.; Mumford, C.; Eglese, R. Investigating the use of metaheuristics for solving single vehicle routing problems with time-varying traversal costs. *J. Oper. Res. Soc.* **2013**, *64*, 34–47. [CrossRef]
19. Cordeau, J.-F.; Ghiani, G.; Guerriero, E. Analysis and Branch-and-Cut Algorithm for the Time-Dependent Travelling Salesman Problem. *Transp. Sci.* **2014**. [CrossRef]
20. Arigliano, A.; Calogiuri, T.; Ghiani, G.; Guerriero, E. A branch-and-bound algorithm for the time-dependent travelling salesman problem. *Networks* **2018**, *72*, 382–392. [CrossRef]
21. Adamo, T.; Ghiani, G.; Guerriero, E. An enhanced lower bound for the time-dependent travelling salesman problem. *Comput. Oper. Res.* **2020**, *113*, 104795. [CrossRef]
22. Melgarejo, P.A.; Laborie, P.; Solnon, C. A time-dependent no-overlap constraint: Application to urban delivery problems. In *International Conference on AI and OR Techniques in Constriant Programming for Combinatorial Optimization Problems*; Springer: Berlin/Heisenberg, Germany, 2015; pp. 1–17.
23. Albiach, J.; Sanchis, J.M.; Soler, D. 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.* **2008**, *189*, 789–802. [CrossRef]
24. Arigliano, A.; Ghiani, G.; Grieco, A.; Guerriero, E.; Plana, I. Time-dependent asymmetric traveling salesman problem with time windows: Properties and an exact algorithm. *Discret. Appl. Math.* **2019**, *261*, 28–39. [CrossRef]
25. Hewitt, M.; Boland, N.; Vu, M.D.; Savelsbergh, M. Solving Time Dependent Traveling Salesman with Time Windows Problems. Available online: http://www.optimization-online.org/DB_FILE/2018/05/6640.pdf (accessed on 30 November 2020).
26. Agustín, M.; Isabel, M.; Juan, J.M.B. An integer programming approach for the time-dependent traveling salesman problem with time windows. *Comput. Oper. Res.* **2017**, *88*, 280–289.
27. Helvig, C.S.; Robins, G.; Zelikovsky, A. The moving-target traveling salesman problem. *Algorithms* **2003**, *49*, 153–174. [CrossRef]
28. Montemanni, R.; Barta, J.; Mastrolilli, M.; Gambardella, L.M. The robust traveling salesman problem with interval data. *Transp. Sci.* **2007**, *41*, 366–381. [CrossRef]
29. Gouveia, L.; Voß, S. A classification of formulations for the (time-dependent) traveling salesman problem. *Eur. J. Oper. Res.* **1995**, *83*, 69–82. [CrossRef]
30. Fox, K.R.; Gavish, B.; Graves, S.C. An n-constraint formulation of the (time-dependent) traveling salesman problem. *Oper. Res.* **1980**, *28*, 1018–1021. [CrossRef]
31. Godinho, M.T.; Gouveia, L.; Pesneau, P. Natural and extended formulations for the time-dependent traveling salesman problem. *Discret. Appl. Math.* **2014**, *164*, 138–153. [CrossRef]
32. Miranda-Bront, J.J.; Méndez-Díaz, I.; Zabala, P. An integer programming approach for the time-dependent TSP. *Electron. Notes Discret. Math.* **2010**, *36*, 351–358. [CrossRef]
33. Picard, J.C.; Queyranne, M. The time-dependent traveling salesman problem and its application to the tardiness problem in one-machine scheduling. *Oper. Res.* **1978**, *26*, 86–110. [CrossRef]
34. Stecco, G.; Cordeau, J.F.; Moretti, E. A branch-and-cut algorithm for a production scheduling problem with sequence-dependent and time-dependent setup times. *Comput. Oper. Res.* **2008**, *35*, 2635–2655. [CrossRef]
35. Wiel, R.J.V.; Sahinidis, N.V. An exact solution approach for the time-dependent traveling-salesman problem. *Nav. Res. Logist.* **1996**, *43*, 797–820. [CrossRef]

36. Ghiani, G.; Guerriero, E. A Note on the Ichoua, Gendreau, and Potvin (2003) Travel Time Model. *Transp. Sci.* **2014**, *48*, 458–462. [CrossRef]
37. Adamo, T.; Ghiani, G.; Guerriero, E. On path ranking in time-dependent graphs. *arXiv* **2020**, arXiv:abs/2009.07588.
38. Aggarwal, C. *Neural Networks and Deep Learning*; Springer: Berlin/Heisenberg, Germany, 2018.
39. MacQueen, J. Some methods for classification and analysis of multivariate observations. In Proceedings of the Fifth Berkeley Symposium on Mathematical Statistics and Probability, Oakland, CA, USA, 21 June–18 July 1965; Volume 1, pp. 281–297.
40. Luo, M.R.; Cui, G.; Rigg, B. The development of the cie 2000 colour-difference formula: Ciede2000. *Color Res. Appl.* **2001**, *26*, 340–350. [CrossRef]