algorithms

MDPI

Article

A New Constructive Heuristic Driven by Machine Learning for the Traveling Salesman Problem

Umberto Junior Mele [1], Luca Maria Gambardella [1] and Roberto Montemanni [2,*]

[1] Dalle Molle Institute for Artificial Intelligence (IDSIA), USI-SUPSI, 6962 Lugano, Switzerland; umberto.junior.mele@usi.ch (U.J.M.); luca.gambardella@usi.ch (L.M.G.)
[2] Department of Sciences and Methods for Engineering, University of Modena and Reggio Emilia, 41121 Modena, Italy
* Correspondence: roberto.montemanni@unimore.it

Abstract: Recent systems applying Machine Learning (ML) to solve the Traveling Salesman Problem (TSP) exhibit issues when they try to scale up to real case scenarios with several hundred vertices. The use of Candidate Lists (CLs) has been brought up to cope with the issues. A CL is defined as a subset of all the edges linked to a given vertex such that it contains mainly edges that are believed to be found in the optimal tour. The initialization procedure that identifies a CL for each vertex in the TSP aids the solver by restricting the search space during solution creation. It results in a reduction of the computational burden as well, which is highly recommended when solving large TSPs. So far, ML was engaged to create CLs and values on the elements of these CLs by expressing ML preferences at solution insertion. Although promising, these systems do not restrict what the ML learns and does to create solutions, bringing with them some generalization issues. Therefore, motivated by exploratory and statistical studies of the CL behavior in multiple TSP solutions, in this work, we rethink the usage of ML by purposely employing this system just on a task that avoids well-known ML weaknesses, such as training in presence of frequent outliers and the detection of under-represented events. The task is to confirm inclusion in a solution just for edges that are most likely optimal. The CLs of the edge considered for inclusion are employed as an input of the neural network, and the ML is in charge of distinguishing when such edge is in the optimal solution from when it is not. The proposed approach enables a reasonable generalization and unveils an efficient balance between ML and optimization techniques. Our *ML-Constructive* heuristic is trained on small instances. Then, it is able to produce solutions—without losing quality—for large problems as well. We compare our method against classic constructive heuristics, showing that the new approach performs well for TSPLIB instances up to 1748 cities. Although *ML-Constructive* exhibits an expensive constant computation time due to training, we proved that the computational complexity in the worst-case scenario—for the solution construction after training—is $O(n^2 \log n^2)$, n being the number of vertices in the TSP instance.

Keywords: traveling salesman problem; machine learning; artificial intelligence; constructive heuristic; hybrid heuristic; reinforcement learning; statistical analysis; complexity theory

Citation: Mele, U.J.; Gambardella, L.M.; Montemanni, R. A New Constructive Heuristic Driven by ML for the TSP. *Algorithms* **2021**, *14*, 267. https://doi.org/10.3390/a14090267

Academic Editors: Frank Werner and Johan Markdahl

Received: 17 August 2021
Accepted: 9 September 2021
Published: 14 September 2021

1. Introduction

The TSP is one of the most intensively studied and relevant problems in the Combinatorial Optimization (CO) field [1]. Its simple definition—despite the membership to the NP-complete class—and its huge impact on real applications [2] make it an appealing problem to many researchers. As evidence of this, the last seventy years have seen the development of extensive literature, which brought valuable enhancement to the CO field. Concepts such as the Held-Karp algorithm [3], powerful meta-heuristics such as the Ant Colony Optimization [4], and effective implementations of local search heuristics such as the Lin-Kernighan-Helsgaun [5] have been suggested to solve the TSP. These contributions,

along with others, have supported the development of various applied research domains such as logistics [6], genetics [7], telecommunications [8] and neuroscience [9].

In particular, during the last five years, an increasing number of ML-driven heuristics have appeared to make their contribution to the field [10,11]. The surge of interest was probably moved by the rich literature, and by the interesting opportunities provided by CO applications. Among the many works recently proposed it is worth mentioning those that have introduced empowering concepts such as the opportunity to leverage knowledge from past solutions [12,13], the ability to imitate computationally expensive operations [14], and the faculty of devising innovative strategies via reinforcement learning paradigms [15].

In light of the new features being brought by ML approaches, we wish to couple these ML qualities with well-known heuristic concepts, aiming to introduce a new kind of hybrid algorithm. The scope is to engineer an efficient interlocking between ML and optimization algorithms, which seeks robust enhancements with respect to classic approaches. Many attempts have been proposed so far, but none of them until now has succeeded in preserving the improvements while scaling up to larger problems. A promising idea to contrast the scaling up issue is to use CLs [16]. A CL identifies a subset of edges that are considered promising for the solution. Using CLs can help the solver to restrict the solution searching space since most of the edges are marked as unpromising and will not be considered in the optimization. Moreover, the employment of CL allows for a *divide et conquer* abstraction, which favors generalization. It can be argued that the generalization issue which emerged in the previous ML-driven approaches is caused mostly by the lack of a proper consideration of the ML weaknesses and limitations [17,18]. In fact, ML is known for having troubles with imbalanced datasets, outliers and extrapolation tasks. Such cases could lead to significant obstacles in achieving good performances with most ML systems. The aforementioned statistical studies were very useful to achieve fundamental insights which allowed our *ML-Constructive* to bypass these weaknesses. More details on these typical ML weaknesses, with our proposed solutions, will be provided in Section 2.2.

Our main contribution is the introduction of the first ML-driven heuristic that actively uses ML to construct partial TSP solutions without losing quality when scaling. The *ML-Constructive* heuristic is composed of two phases. The first phase uses ML to identify edges that are very likely to be optimal, the second completes the solution by a classic heuristic. The resulting overall heuristic shows good performance when tested on representative instances selected from the TSPLIB library [19]. The instances considered present up to 1748 vertices, and surprisingly *ML-Constructive* exhibits slightly better solutions on larger instances rather than on smaller ones, as shown in the experiments. Despite the fact that good results are shown in terms of quality, our heuristic presents an unappealing large constant computation time for training in the current state of the implementation. However, we prove that for the creation of a solution a number of operations bounded by $O(n^2 \log n^2)$ is required after training (which is executed only once). *ML-Constructive* learns exclusively using local information, and it employs a simple ResNet architecture [20] to recognize some patterns from the CLs through images. The use of images, even if not optimal in terms of computation, allowed us to plainly see the input of the network and to get a better understanding of the internal processes in the network. We have finally introduced a novel loss function to help the network to understand how to make a substantial contribution when building tours.

The TSP is formally stated in Section 1.1, and a literature review is presented in Section 1.2. The concept of constructive heuristic is described in detail in Section 2.1 while statistical and exploratory studies on the CLs are spotlighted in Section 2.2. The general idea of the new method is discussed in Section 2.3, the *ML-Constructive* heuristic is explained in Section 2.4 and the ML model with the training procedure is discussed in Section 2.5. To conclude, experiments are presented in Section 3, and conclusions are stated in Section 4.

1.1. The Traveling Salesman Problem

Given a complete graph $G(V, E)$ with n vertices belonging to the set $V = \{0, \ldots, n-1\}$, and edges $e_{ij} \in E$ for each vertex $i, j \in V$ with $i \neq j$, let c_{ij} be the cost for the directed edge e_{ij} starting from vertex i and reaching vertex j. The objective of the Traveling Salesman Problem is to find the shortest possible route that visits each vertex in V exactly once and creates a loop returning to the starting vertex [1].

The [21] formulation of the TSP is an integer linear program describing the requirements that must be met to find an optimal solution to the problem. The variable x_{ij} defines if the optimal route found picks the edge that goes from vertex i to vertex j with $x_{ij} = 1$, if the route does not pick such edge then $x_{ij} = 0$. A solution is defined as a matrix X with entries x_{ij} and dimension $n \times n$. The objective function is to minimize the route cost, as shown in Equation (1).

$$\min \sum_{i=0}^{n-1} \sum_{j=0, j\neq i}^{n-1} c_{ij}\, x_{ij} \tag{1}$$

$$\sum_{i=0, i\neq j}^{n-1} x_{ij} = 1, \qquad j = 0, \ldots, n-1 \tag{2}$$

$$\sum_{j=0, j\neq i}^{n-1} x_{ij} = 1, \qquad i = 0, \ldots, n-1 \tag{3}$$

$$\sum_{i\in Q} \sum_{j\in Q, j\neq i} x_{ij} \leq |Q| - 1, \qquad \forall Q \subset \{0, \ldots, n-1\},\ n > |Q| \geq 2 \tag{4}$$

$$x_{ij} \in \{0, 1\}, \qquad i, j = 0, \ldots, n-1, \quad i \neq j \tag{5}$$

There are the following constraints: each vertex is arrived at from exactly one other vertex (Equation (2)), each vertex is a departure to exactly one other vertex (Equation (3)) and no inner-loop between vertices for any proper subset Q is created (Equation (4)). The constraints in Equation (4) prevent that the solution X is the union of smaller tours. To conclude, each edge x_{ij} in solution must be not fractional (Equation (5)). We point out that the graphs used in this work are symmetric and placed in a two-dimension space.

A CL with cardinality k for vertex i is defined as the set of edges e_{ij}, with $j \in CL[i]$, such that the vertices j are the closest k vertices to vertex i. Note that for each vertex there is a CL, and for each CL there are at most two optimal edges.

1.2. Literature Review

The first constructive heuristic documented for the TSP is the Nearest Neighbor (NN), which is a greedy strategy that repeats the rule *"take the closest vertex from the set of unvisited vertices"*. This procedure is very simple, but it is not very efficient. While the NN is choosing the best vertex to join the solution, the Multi-Fragment (MF) [22,23] and the Clarke-Wright (CW) [24] are alternatives to add the most promising edge in the solution. Note that the NN grows a single fragment of the tour by adding the closest vertex to the fragment extreme considered during the construction. On the other hand, MF and CW grow, join and give birth to many fragments of the final tour [25]. The approaches using many fragments show superior quality performances, and also come up with very low computational costs.

A different way of constructing TSP tours is known as insertion heuristic, such as the Furthest-Insertion (FI) [23]. These approaches iteratively expand the tour generated by the previous iteration. Considering that—at the mth iteration—the insertion heuristic has created a feasible tour with $m + 1$ vertices (iteration one starts with two vertices) belonging to the inserted edges subset $\hat{V}_m \subset V$. At iteration $m + 1$, the expansion is carried out, and a new vertex, e.g., vertex j, is inserted into the subset $\hat{V}_{m+1} = \hat{V}_m \cup \{j\}$. To preserve the feasibility of the expanded tour, one edge is removed from the previous tour and two edges are added to connect the released vertices to the new vertex j. The removal is done in such a way that the lowest cost for the new tour is achieved.

In case the insertion policy is *Furthest*, the new inserting vertex is always the farthest from all the vertices belonging to the current tour. Once the last iteration is reached, a complete feasible tour passing through each vertex in V has been constructed. For further details about the computational complexity and the functioning of these broadly used constructives (MF, CW and FI) we suggest chapter six of Gerhard Reinelt's book [26].

Initially, the exploration in the literature produced to introduce ML concepts was about a setup similar to the NN. Systems such as pointer networks [27] or graph-based transformers [12,13,28] were engaged to select the next vertex in the NN iteration. These architectures were engaged to predict values for each valid departure edge of the extreme considered on the single fragment approach. Then, the best choice (next vertex) according to these values was added to the fragment. These systems were applying stochastic sampling as well, so they could produce thousands of solutions in parallel using GPU processing. Unfortunately, these ML approaches failed to scale, since all vertices were given as input to the networks, and TSP instances with different dimensions exhibit inconsistent intrinsic features.

To attempt to overcome the generalization problem, several works proposed systems arguably claimed to be able to generalize [16,29,30]. Results however showed that the proposed *structure2vec* [29] and Graph Pointer Network [30] keep their performances close to the FI [23] up to 250 vertices instances, then they lose their ability to generalize. The model called Att-GCN [16] is used to pre-process the TSP by initializing CLs. Even if solutions are promising in this case, the ML was not actively used to construct TSP tours. Furthermore, no comparisons with other CL constructors as: POPMUSIC [31], Delunay Triangolarization [32], minimum spanning 1-tree [1] and a recent CL constructor driven by ML [33] were provided in the paper.

The use of Reinforcement Learning (RL) to tackle TSP was proposed as well [34]. The ability to learn heuristics without the use of optimal solutions is very appealing. These architectures were trained just via the supervision of the objective function shown in Equation (1). Several RL algorithms were applied so far to solve TSP (and CO problems in general [35]). The actor-critic algorithm was employed in [36]. Later, ref. [13] used the actor-critic paradigm with an additional reward defined by the easy to compute minimum spanning tree cost (resembling a TSP solution, see [37]). Moreover, the Q-learning framework was used by [29], the Monte Carlo Tree Search was employed by [16] and Hierarchical RL applied in [30]. It is worth mentioning that [38] explicitly advised rethinking about the generalization issues to solve TSP employing ML, suggesting that a more hybrid approach was necessary. In Miki [39,40] was proposed for the first time the use of image processing techniques to solve the TSP. The choice—even if not obvious in terms of efficiency—has the advantage to get a better understanding of internal network processes.

2. Materials and Methods

For the sake of clarity, materials and methods employed in this work have been divided into subsections. In Section 2.1, constructive heuristics, based on fragments growth, are reviewed with some examples. The statistical study and the main intuitions behind our heuristic are presented in Section 2.2. The general idea of the *ML-Constructive* heuristic is described in Section 2.3, the overall algorithm with its complexity is demonstrated in Section 2.4. The ML system, the image creation process and the training procedure are explained in detail in Section 2.5.

2.1. Constructive Heuristics

Constructive heuristics are employed to create TSP solutions from scratch, when just the initial formulation described in Section 1.1 is available. They exploit intrinsic features during the solution creation, generally provide quick solutions with modest quality, and exhibit low polynomial computational complexity [1,24,41].

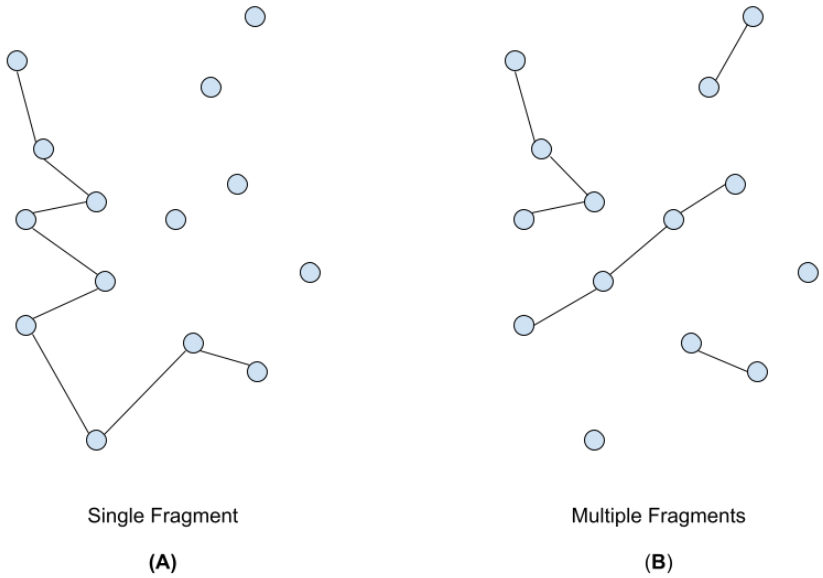

Figure 1. (**A**) Single fragment constructor that operates similarly to NN. (**B**) Constructor that growths multiple fragments similarly to MF and CW.

In this paper, we further develop constructive heuristics. Particularly, those that take their decisions on edges are addressed. These approaches grow many fragments of the tour, in opposition to NN that grows just a single fragment (Figure 1). Since at each addition the procedure must avoid inner-loops and no vertex can be connected with more than two other vertices, they take an extra effort concerning the NN to preserve the TSP constraint. However, the computational time needed to construct a tour remains limited [26].

As introduced by [42,43], two main choices are required to design an original constructive heuristic driven by edge choices: the order for the examination of the edges, and the constraints that ensure a correct addition. Note that to construct an optimal TSP solution the examination order of the edges must be optimal as well (there are more optimal orders). So, theoretically, the objective of an efficient constructive is to examine the edges in the best possible order.

The examination order is arranged according to the relevance that each edge of the instance exhibits. The relevance of an edge is related to the probability that we expect that such an edge is in the optimal solution. The higher is the relevance the earlier that edge should be examined. Different strategies are possible to measure the edges' relevance, the most famous ones are the MF and the CW policies. The MF relevance is expressed by the cost values, the smaller is the cost, the higher is the probability of being added. Instead, the CW relevance depends on the saving value, which was designed on purpose to rethink the examination order. Saving is the gain obtained when rather than passing through a hub node h, the salesman uses the straight connection between vertex i and vertex j. Note that the hub vertex is chosen to exhibit the shortest Total Distance (TD) from the other vertices. Equation (6) describes the formulas used to find the hub, while Equation (7) shows the function employed to compute the saving values.

$$h = \arg\min_{i \in V} \; TD[i], \qquad TD[i] = \sum_{j=0}^{n-1} c_{ij}, \qquad \forall i \in V \tag{6}$$

$$s_{ij} = c_{ih} + c_{hj} - c_{ij}, \qquad \forall i, j \in V, \text{ with } i \neq j \tag{7}$$

The second important choice is to define simple constraints that ensure correct additions. The edge's addition checker is the algorithm the allows to add feasible edges at each iteration. For both approaches (MF and CW), it is checked that the examined edge does not exhibit extremes vertices with already two connections (Equations (2) and (3)),

and that does not create inner-loops (Equation (4)) with the current partial solution. *Tracker* is called the subroutine that checks if the examined edge can create an inner-loop, and it uses a quadratic number of operations for the worst-case scenario in our implementation (Appendix A). Note that there exist more efficient data structures and algorithms for the tracker subroutine that even runs in $O(n \log n)$ [43].

2.2. Statistical Study

As mentioned earlier, the generalization issues of ML approaches are likely caused by a poor consideration of well-known deep learning weaknesses [17,18,38]. It is known that dealing with imbalanced datasets, outliers and extrapolation tasks can critically affect the overall performances of an ML system. So one of the main reasons for using ML for TSP is to reduce the number of wrong forecasts during the construction of the solution.

An outlier arises when a data point differs significantly from other observations, outliers can cause problems during back-propagation [44] and in their pattern recognition [45]. Finally, with extrapolation it is mean the phenomenon that occurs when a learned system is required to operate beyond the space of known training examples, since it wants to extend the intrinsic features of the problem to similar but different tasks [46].

To overcome the aforementioned problems, we suggest supporting the ML with an optimization heuristic. We instruct the model to act as a decision-taker, and we engineer to place it in a context that allows it to act confidently. We emphasize the significance of designing a good heuristic that avoids gross errors, e.g., by omitting imbalanced class skews and outlier points. Choosing wisely a context for the ML that does not change too frequently during the algorithm iterations, can help it to deal with extrapolation tasks as well.

We addressed the challenge by introducing two operational components: the use of a subroutine and the employment of ML as a decision-taker for the solution construction. The subroutine consists of the detection of optimal edges from the elements of a CL. As stated in Section 1.1, CL identifies the most promising edges to be part of the optimal solution. For instance, [47] proved that just around $30 \cdot n$ of the edges need to be taken into account by an optimal solver for large instances with more than a thousand points. The use of CL with ML was firstly suggested by [16]; but, as mentioned, they employed ML to initialize CL rather than constructing tours.

To understand the decision-taker task, an exploratory study was carried out to check the distribution of the optimal edges within the CLs. It was observed that after sorting the edges in the CL from the shortest to the longest, the occurring of an optimal edge was not uniform concerning the positions in the sorted CL, but followed a logarithmic distribution, as shown in Figure 2. Such a pattern unfortunately revealed a severe class distribution skew for some positions. In fact, some positions displayed the presence of optimal edges much more frequently than other positions.

Figure 2 also shows the rate of optimal edges found for each position. Note that an optimal edge occurrence arises when the optimal tour passes through the edge in the position taken into consideration by the CL. This study emphasizes the relevance of detecting when the CL's shortest edge is optimal since about 88.6% for the evaluation data-set and 86.7% for the test data-set of the times these edges belong to the optimal tour. However, it reveals as well that detecting with ML when the shortest edge is not optimal is a hard task due to the over-represented situation.

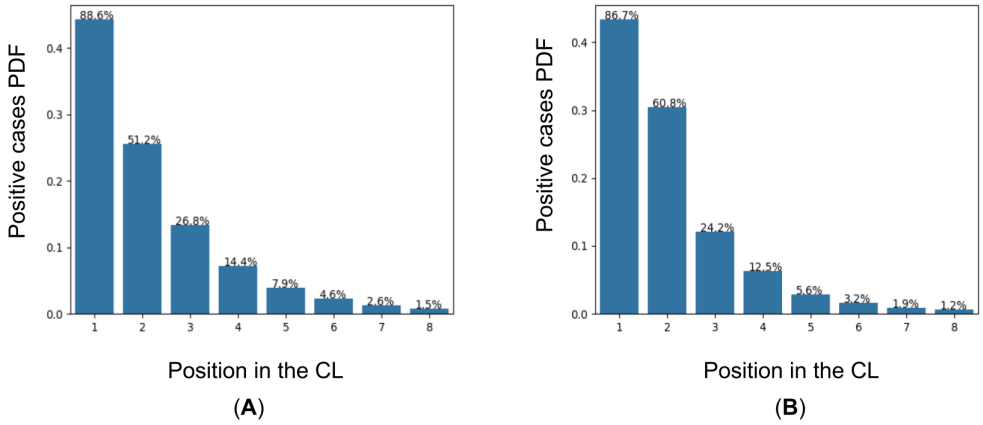

Figure 2. Empirical Probability Density Function (PDF) showing the optimal edge behavior in relation to the position in the CL. Over each bar is shown the rate of optimal edge occurrence for each considered position. (**A**) Evaluation data-set. (**B**) TSPLIB data-set.

Instead, considering the second shortest edge, a balanced scenario is observed. About half of the occurrences are positive and the other half is negative for both data sets. On the other hand, the rapid growth of under-represented positions can be observed from the third position onwards. Note that up to the fifth position the under-representation is not too severe, and imbalanced learning techniques could make their contribution to infer some useful patterns [48]. From the sixth on instead, the optimal occurrences are too rare to be able to recognize any useful pattern, even if these positions could be interpreted as very useful ones in terms of construction. Considering the rate of optimal edges shown in Figure 2, the sum of optimal occurrences for the first five positions in the CL represents about 95% of the total optimal edges available. Therefore, the selection of such a subset of edges is promising in regards to ML pattern recognition, in such a way as to avoid all the under-represented scenarios.

The empirical probability density functions (PDF) shown in Figure 2 were computed using 1000 uniform random euclidean TSP instances—sampled in the unit side square with a total number of vertices varying uniformly between 500 and 1000—for the evaluation data-set and a representative selection of TSPLIB instances as test data-set. The latter data-set was select in such a way that all the instances available in the TSPLIB library with a total number of vertices varying between 100 and 2000 were included. Furthermore, these instances were required to be stacked in a two-dimensional space as well. The optimal solutions were computed with the Concorde solver [49] for both cases.

After that the most promising edges were selected, a relevant choice to be made in our heuristic is the examination order of these edges. As mentioned, edges selected in earlier stages of the construction exhibit higher probabilities to be accepted regarding the later ones. To explore the effectiveness of different strategies, we tested the behavior of classic constructive solvers such as MF, FI and CW (Figure 3). Using the representative TSPLIB instances [19], these solvers were investigated on some classic metrics such as the TPR, the FPR, the accuracy and the precision. Considering each constructive solver as a predictor, each position in the CL as a sample point, and the optimal edge positions as the actual targets to be predicted. For each CL, there are two position targets and two positions predicted.

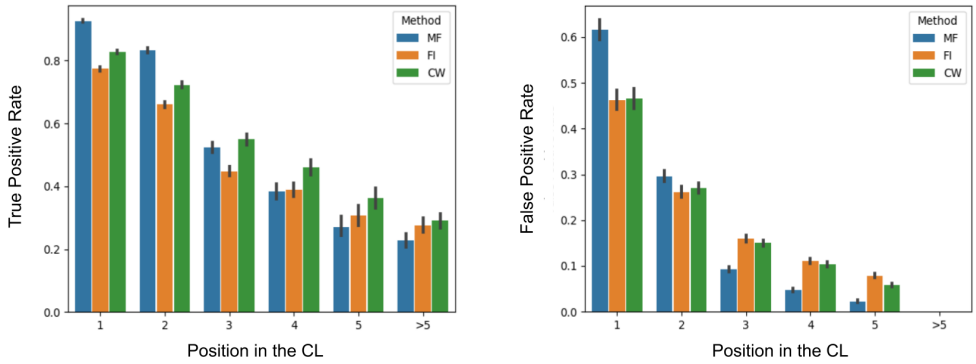

Figure 3. TPR (**A**) and FPR (**B**) comparison for MF, FI and CW heuristics. The first five positions in the CL are considered separately, while all the others are shown in the *>5* bars.

A true-positive occurs when the predicted edge is also optimal, a false-positive instead occurs if the predicted edge is not optimal. Note that avoiding false-positive cases is crucial since they block other optimal edges in the process. To take care also of this aspect, the precision of the predicting heuristic was considered in Table 1 as well. Let D_p be the dataset of all the edges available in the p positions for each CL. If the predictor truly finds an optimal edge in the observation i, the variable TP_i will be equal to one, otherwise, it is zero. Similarly, it is for the false-positive FP_i variable. Note that a positive (P_i) or negative (N_i) observation occur when the observation is optimal or not, respectively,and the frequency of these events varies according to the p position. Hence, studying the predictor performances by position is important since each position has different importance during the solution construction and for the optimal frequency.

$$TPR = \frac{\sum_{i \in D_p} \mathrm{TP}_i}{\sum_{i \in D_p} \mathrm{P}_i}, \qquad FPR = \frac{\sum_{i \in D_p} \mathrm{FP}_i}{\sum_{i \in D_p} \mathrm{N}_i} \tag{8}$$

Table 1. TPR, FPR, accuracy and precision comparison across several positions and methods.

Position	Method	TPR	FPR	Accuracy	Precision
1	MF	92.57%	61.65%	85.20%	90.52%
	FI	77.39%	46.26%	74.18%	91.41%
	CW	82.79%	46.56%	78.80%	91.88%
2	MF	83.21%	29.57%	78.19%	81.30%
	FI	66.00%	26.20%	69.06%	79.56%
	CW	72.29%	27.01%	72.57%	80.53%
3	MF	52.41%	9.23%	81.59%	64.15%
	FI	44.80%	15.99%	74.62%	46.87%
	CW	55.03%	15.04%	77.79%	53.54%
4	MF	38.47%	4.79%	88.15%	53.28%
	FI	38.96%	11.09%	82.70%	33.30%
	CW	45.99%	10.40%	84.18%	38.59%
5	MF	27.12%	2.27%	93.75%	41.65%
	FI	30.59%	7.88%	88.65%	18.82%
	CW	36.20%	5.75%	90.98%	27.35%
>5	MF	22.72%	0.01%	99.98%	13.94%
	FI	27.55%	0.03%	99.97%	9.60%
	CW	29.01%	0.02%	99.98%	14.71%

To get a better look at the results shown in Figure 3, Table 1 emphasizes the values obtained during the experiment. MF exhibits a higher TPR for shorter edges as expected, while CW performs better with longer edges. However, less obvious is MF's higher FPR in the first position. The latter fact brings attention to the importance of decreasing the FPR for the most frequent position. Note that the CW's precision is higher concerning the other ones for the first position, which can be read as the main reason why CW comes up with better TSP solutions than MF. So, one of the main reasons for using ML for TSP is to reduce the FPR during the construction of the solution.

2.3. The General Idea

In light of the statistical study presented in Section 2.2, we propose a constructive heuristic called *ML-Constructive*. The heuristic follows the edge addition process (see MF and CW in Section 2.1) extended by an auxiliary operation that asks the ML to agree for any attaching edge during a first phase. The goal is to avoid as much as possible adding bad edges in the solution while allowing the addition of promising edges which are considered auspicious by the ML model. We emphasize that our focus is not on the development of highly efficient ML architectures, but rather on the successful interaction between ML and optimization techniques. Therefore, the ML is conceived to act as a decision-taker and the optimization heuristics as the texture of the solution building story. The result is a new hybrid constructive heuristic that succeeds in scaling up to big problems while keeping the improvements achieved thanks to ML.

As aforementioned, the ML is exploited just in situations where the data do not suggest underrepresented cases, and since about 95% of the optimal edges are connections with one of the closest five vertices of a CL, only such subset of edges is initially considered to test the ML performances. Recall that it is a common practice to avoid employing ML in the prediction of rare events. Instead, it is commonly suggested to apply ML preferably in cases where a certain level of pattern recognition can be confidently detected. For this reason, our solution is designed to construct TSP tours in two phases. The first employs ML to construct partial solutions (Figure 4). While the second uses the CW heuristic to connect the remaining free vertices and complete the TSP tour.

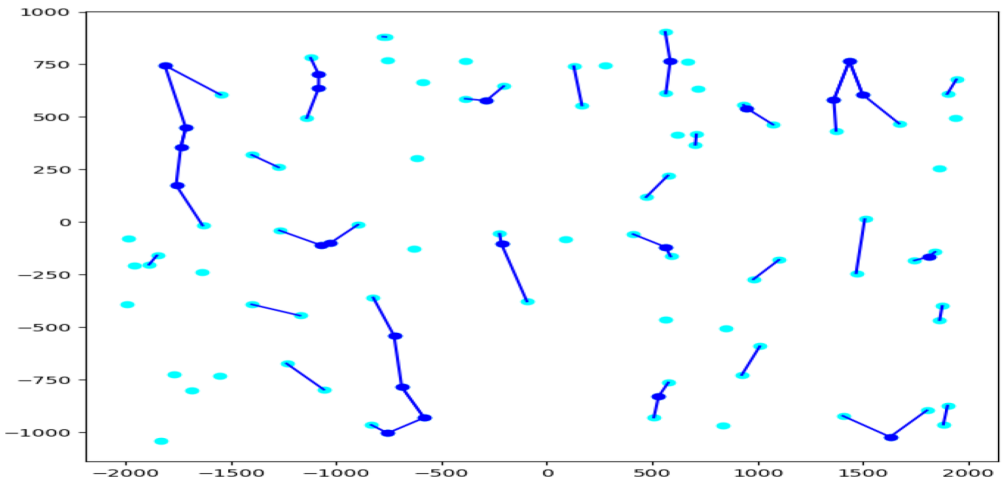

Figure 4. First phase partial solution constructed with the ML predictions. Vertices in light blue are free for the second phase of *ML-Constructive*. The instance is the KroA100 from the TSPLIB collection.

Initially, during the first phase, considered most likely edges to be found in the optimal tour are collected in the list of promising edges L_P. To appropriately choose these edges, several experiments were carried out and results are shown in Table 2. The strategy used to build the promising list was to include the edges of the first m vertices of each CL, considering the m value ranging from 1 to 5. The ML was in charge of predicting whether the edges under consideration in L_P were in the optimal solution or not. Experiments were

handled adopting the same ResNet [20] architecture and procedure explained in Section 2.5, but the ML was trained on different data to be consistent with the m tested value.

Table 2. Comparison on TPR, FPR and their difference for several choices of the L_P list.

Closest m	TPR	FPR	TPR-FPR
1	100.00%	100.00%	0.00%
2	53.91%	13.70%	40.21%
3	30.97%	1.64%	29.33%
4	31.00%	1.50%	29.50%
5	30.66%	1.30%	29.36%

Several classic metrics were compared for the different m: the True Positive Rate (TPR), the False Positive Rate (FPR), and their difference [50]. Please note that ML objectives are to keep the FPR small, meanwhile to obtain good results in terms of TPR. Keeping a small FPR ensures that during the second phase the *ML-Constructive* has an higher probability in detecting good edges, meanwhile with a high TPR the search space for the second phase is hopefully reduced (Appendix B).

In terms of TPR, it seems to be the best choice to include just the shortest edge in L_P ($m = 1$), but by checking the FPR in Table 2 it becomes obvious that the ML has learned to predict almost always an agreement for $m = 1$. Therefore, such a behavior is undesirable and it leads to a high FPR, and hence to worse solutions during the second phase. However, if the difference between TPR and FPR is taken into account, the best arrangement is when the first two shortest edges are put into the list ($m = 2$). Although other arrangements might show to be effective as well, the selection through positions in the CLs and the selection of the first two shortest edges in each CL are proven to be efficient by the results. Recall that too many edges in L_P can be confusing since outliers and classes with severe distribution skew can appear. For example, detecting optimal edges from the fourth position onwards is very difficult since they are very under-represented (Figure 2), and the creation of images connecting edges that are in the fourth position in their CL is very uncommon—causing outliers in the third channel (Section 2.5).

After engineering the promising list structure, L_P is sorted according to a heuristic that seeks to anticipate the processing of good edges. It is crucial to find an effective sorting heuristic for the promising list since the order of it affects the learning process and the *ML-Constructive* algorithm as well. An edge belonging to the optimal tour and being straightforwardly detected by the ML model is regarded as good. Therefore, for simplicity, in this work, the list is sorted by edge's position in the CL and cost length, but other approaches could be propitious, perhaps using ML. Note that, as repeatedly mentioned, the earlier examination of the most promising edges increases the probability to find good tours employing the multiple fragment paradigm (Appendix B).

At this point, the edges belonging to the sorted promising list are drawn in images and fed to the ML one at a time. If the represented edge meets the TSP constraints considering the partial solution found at the current iteration, the ML system will be challenged to detect if the edge is in the optimal solution. If the ML agrees with a given level of confidence, the heuristic will add the edge to the solution. Assuming that some CLs information provides enough details to detect common patterns from previous solutions, the images represent just a small subset of vertices given by the CLs of each edge extremes of the edge that is processed. The partial solution visible in such local context and available up to the insertions made by moving through the promising list is represented as well.

Once all the edges of the promising list have been processed, the second phase of the algorithm will complete the tour. Initially, it detects the remaining free vertices to connect (Figure 4), then it connects such vertices employing the CW. Note that CW usually captures the optimal long edges better than MF, as emphasized in Figure 3 and Table 1. Therefore CW represents a promising candidate solver to connect the remaining free fragment extremes of the partial solution into the final tour. However, other arrangements

employing local searches or meta-heuristics may be considered promising as well even if more time-consuming [51].

2.4. The ML-Constructive Algorithm

The *ML-Constructive* starts as a modified version of MF, then concludes the tour exploiting the CW heuristic. The ML model behaves consequently as a glue since it is crucial to determine the partial solution available at the switch between solvers.

The list of promising edges L_P and the confidence level of the ML decision-taker are critical specifications to set in the heuristic before than it runs. The reasons behind our promising list building choices were widely discussed in Section 2.3. While the confidence level is used to handle the exploitation vs exploration trade-off. It consists of a simple threshold applied to the predictions made by the ML system. If the predicted probability that validates the insertion is greater than such threshold, then the insertion is applied. The value of 0.99 has been verified to provide good results on tested instances. Since, lower values increase the occurrence of false-positive cases, thus leading to the inclusion of edges that are not optimal. On the other hand, higher values of it decrease the occurrence of true-positive cases, hence increasing the challenge of the second phase.

The overall pseudo-code for the heuristic is shown in Algorithm 1. Firstly, the CLs for each vertex in the instance are computed (line 2). We noticed that considering just the closest thirty vertices for each CL was a good option. As mentioned, just the first two connections are considered in L_P, while the other vertices are used to create the local context in the image. The CL construction takes a linear number of operations for each vertex, and the overall time complexity for constructing it is $O(n^2)$. Since finding the nearest vertex of a given vertex it takes linear time, the search for the second nearest takes the same time (after removing the previous from the neighborhood). So on until the thirtieth nearest vertex is found. As only the first thirty edges are searched, the operation can be completed in linear time. Then, promising edges are inserted in L_P (line 3), then repeating edges are deleted to avoid unnecessary operations (line 4). The list is sorted according to the position in the CL and the cost values (line 5). All the edges that are the first nearest will be found first and sorted according to c_{ij}, then the second nearest and so on. Since only the first two edges for each CL can be in the list, the sorting task is completed in $O(n \log n)$. Several orders for L_P were preliminary tested—e.g., employing descending cost values or even savings to sort the edges in L_P—but experiments suggested that sorting the list according to ascending cost values is the best arrangement.

The first phase of *ML-Constructive* takes part. An empty solution $X = \bar{0}$ is initialized (line 6), and following the order in L_P a variable l is updated with the edge considered for the addition (line 7). At first, l is checked to ensure that the edge complies the TSP constraints (Equations (2)–(4)). Then the ML decision-taker is queried to confirm the addition of the edge l. If the predicted probability is higher than the confidence level, the edge is added to the partial solution (lines 11 and 14). To evaluate the number of operations that this phase consumes, we must split the task according to the various sub-routines which are acted at each new addition in the solution. The "if" statements (lines 9, 13, 23 and 26) check that the constraints (2) and (3) are complied.

They verify that both extremes of the attaching edge l exhibit at most one connection in the current solution. The operation is computed with the help of hash maps, and it takes constant time for each considered edge l. The tracker verification (lines 10 and 24) ensures that l will not create an inner-loop (Equation (4)). This sub-routine is applied only after it is checked that both extremes of l have exactly one connection, each in the current partial solution. It takes overall $O(n^2)$ operations up to the final tour (proof in Appendix A).

Once all the constraints of the TSP have been met, the edge l is processed by the ML decision-taker (lines 11 and 14). Even if time-consuming, such sub-routine is completed in constant time for each l. Initially, the image depicting the l edge and its local information is created, then it is given as input to the neural network. To create the image, the vertices of the CLs and the existing connections in the current partial solution must be retrieved.

Hash maps are used for both tasks, and since the image can include up to sixty vertices, this operation takes a constant amount of time for each l edge in L_P. The size of the neural network does not vary with the number of vertices in the problem as well but remains constant for each edge in L_P.

To complete the tour, the second phase starts by identifying the hub vertex (line 15). It considers all the vertices in the problem (free and not), following the rule explained in Equation (6). Free vertices are selected from the partial solution, and edges connecting such vertices are inserted in the difficult edges list L_D (line 16). The saving for each edge in L_D is computed (line 17), and the list is sorted according to these values (line 18) in $O(n^2 \log n^2)$. At this point (lines 20 to 27), the solution is completed employing the classical multiple fragment steps, which are known to be $O(n^2)$ [41].

Therefore, the complexity of the worst-case scenario for the *ML-Constructive* is:

$$O(n + n \log n + n^2 + n^2 \log n^2) = O(n^2 \log n^2) \tag{9}$$

Note that to complete the tour we proposed the use of CW, but rather hybrid approaches that also use some sort of exhaustive search could be very promising as well, although more time-consuming.

Algorithm 1 ML-Constructive.

Require: TSP graph $G(V, E)$
Ensure: a feasible tour X
1: **procedure** ML-CONSTRUCTIVE($G(V, E)$)
2: create CL for each vertex
3: insert the shortest two vertices for each CL into L_P
4: remove from L_P all duplicate edges in their higher positions
5: sort L_P according to the position in the CL and the ascending costs $c_{i,j}$
6: $X = \bar{\mathbf{0}}$
7: **for** l in L_P **do**
8: select the extreme vertices i, j of l
9: **if** vertex i **and** vertex j have exactly one connection each in X **then**:
10: **if** l do not creates a inner-loop **then**:
11: **if** the ML agrees the addition of l **then**: $x_{i,j} = 1$
12: **else**
13: **if** vertex i **and** vertex j have less than two connections each in X **then**:
14: **if** the ML agrees the addition of l **then**: $x_{i,j} = 1$
15: find the hub vertex h
16: select all the edges that connects free vertices and insert them into L_D
17: compute the saving values with respect to h for each edge in L_D
18: sort L_D according to the descending savings $s_{i,j}$
19: $t = 0$
20: **while** the solution X is not complete **do**
21: $l = L_D[t], \quad t = t + 1$
22: select the extreme vertices i, j of l
23: **if** vertex i **and** vertex j have less than two connections each in X **then**:
24: **if** l do not creates a inner-loop **then**: $x_{i,j} = 1$

2.5. The ML Decision-Taker

The ML decision-taker validates the insertions made by the *ML-Constructive* during the first phase. Its scope is to exploit the ML pattern recognition to increase the occurrences of finding good edges while reducing them for the bad edges.

Two data sets were specifically created to fit the ML decision-taker; the first was used to train the ML system while the second was to evaluate and choose the best model. The training data-set is composed of 38,400 instances uniformly sampled in the unit-sided

square. The total number of vertices for instance n ranges uniformly from 100 to 300. On the other hand, the evaluation data-set is composed of 1000 instances uniformly sampled from the unit-sided square. The total number of vertices varies in this case from 500 to 1000. The data-set has been used to create the results in Table 2. The optimal solutions were found (in both cases) using the Concorde solver [49]. The creation of the training instances and their optimal tours took about 12 hours on a single CPU thread, while a total time of 24 CPU hours was needed for the evaluation data-set creation (since it includes instances of greater size). Note that, in comparison with other approaches that use RL, good results were achieved here even though we used far fewer training instances [13].

To get the ML input ready, the promising list L_P was created for each instance in the data sets. In case $m = 2$ (best scenario), the two shortest edges of each CL were inserted into the list. To avoid repetitions, edges that occur several times in the list were inserted just once at the shortest available position. For example, if edge e_{ij} is the first shortest edge in CL$[i]$ and the second shortest edge in CL$[j]$, it will only occur in L_P once such as the first position for vertex i. After that all promising edges had been inserted into L_P, the list was sorted. Note that the list can contain at most $m \times n$ items in it. An image with a dimension of $96 \times 96 \times 3$ pixels was created for each edge belonging to L_P. Three channels (red, green, and blue) were set up to provide the information used to feed into the neural network. Each channel depicts some information inside a square with sides of 96 pixels each (Figure 5). The first channel of the image (red) shows each vertex in the local view, the second one (green) shows the edge l considered for the insertion with its extremes and the third one (blue) shows all the existing fragments currently in the partial solution and visible from the local view drawn in the first channel.

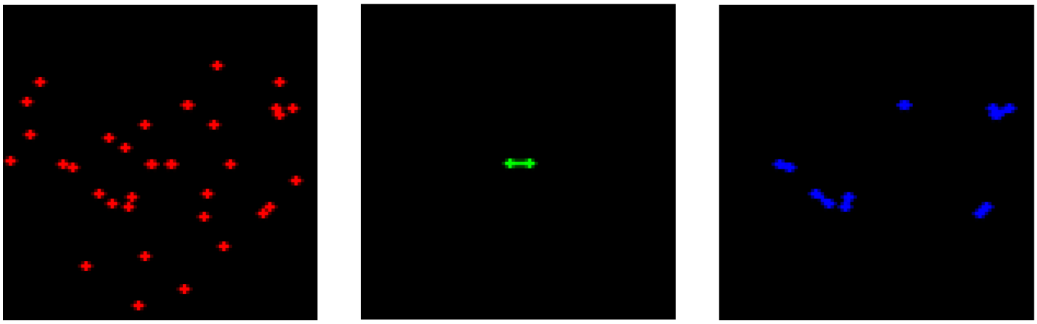

Figure 5. Example of input image. The vertices in the local view are in the red channel, the l edge is drawn in green channel, while the edges in the partial solution are in the blue channel.

As mentioned, the local view was formed by merging the vertices belonging to the CLs of each extreme of the inserting edge. These vertices were collected and their positions normalized to fill the image. The normalization required having the middle of the inserting edge l such as the image center, whereas all the vertices visible in the local view were interior to a virtual sphere inscribed into the squared image, such that the maximum distance between the image center and the vertices in the local view was less than the radius of such sphere. The scope of the normalization was to keep consistency among the images created for the various instances.

The third channel was concerned with giving a temporal indication to aid the ML system in its decision. Representing those edges that had been inserted during the previous stages of the *ML-Constructive*, this information gave a helpful hint in the interpretation of which edges the final solution needs most. Two different policies were employed in the construction of it: the optimal policy (offline) and the ML adaptive policy (online). The first used the optimal tour and the L_P order to create this channel (just on training), while the second one used the ML previous validations (train and test).

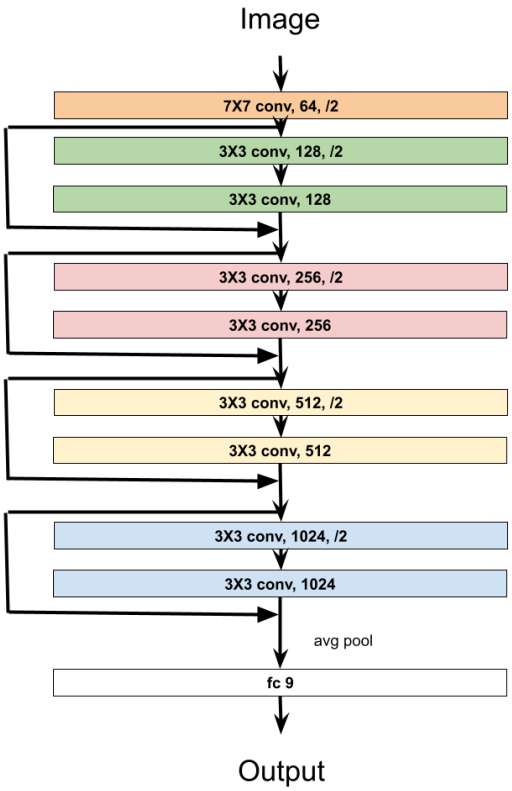

Figure 6. ResNet10.

A simple ResNet [20] with 10 layers was adopted to agree on the inclusion of the edges into the solution. The choice of the model is motivated by the easiness that image processing ML models show on the understanding of the learning process. The architecture is shown in Figure 6. There are four residual connections, containing two convolutional layers each. The first layer in each residual connection is characterized by a stride equal to two (/2). As usual for the ResNet the kernels are set to 3×3, and the number of features increases by multiplying by 2 at each residual connection, to balance the downscale of the images. The output is preceded by a fully connected layer with 9 neurons (fc) and by an average pool operation (avg pool) that shrinks the information in a single vector with 1024 features. For additional details we refer to [20]. The model is very compact, with the scope of avoiding the computational burden and other complexities.

The output of the network is represented by two neurons. One neuron is predicting the probability that the considered edge l is in the optimal solution, while the other is predicting if the edge is not optimal. The sum of both probabilities is equal to one. The choice of using two neurons as output instead of just one is due to the exploitation of the Cross-Entropy loss function, which is recommended to train classification problems. In fact, this loss penalizes especially those predictions that are confident and wrong. The network will know if the inserting edge l is optimal or not during train, while the ResNet should predict the probability that the edge is optimal during the test.

To train the network two loss functions were jointly utilized: the Cross-Entropy loss (Equation (10)) [52] and a reinforcement loss (Equation (11)) which was developed specifically for the task at hand. Initially, the first loss is employed alone up to convergence (about 1000 back-prop iterations), then the second loss is also engaged in the training. At each iteration of back-propagation the first loss updates the network firstly, then (after 1000 iterations) the second loss updates the network as well. The gradient of the second loss function is approximated by the REINFORCE algorithm [53] with a simple moving average with 100 periods used as a baseline.

$$\text{loss}_1 = -\mathop{\mathbb{E}}_{p(x_l)}\left[\log q_\theta(x_l)\right] \tag{10}$$

$$\text{loss}_2 = -\mathop{\mathbb{E}}_{q_\theta(x_l)}\left[T(x_l) - F(x_l)\right] \tag{11}$$

In Equations (10) and (11), x_l is the image of the inserting edge l, the function identifying whether l is optimal is accounted as p, while q_θ is the ResNet approximation to it. Moreover, the T function returns one if the prediction made by q_θ is true (TP or TN), and zero otherwise. While the F function returns one if the prediction is false. Note that the second loss exhibit an expected value with respect to q_θ measure, since the third channel is updated using the ML adaptive policy (online), while the first loss uses the optimal policy (offline). The introduction of a second loss had the purpose of increasing the occurrences of true-positive while decreasing the false-positive cases. Note that it employs the same policy that will occur during the *ML-Constructive* test run.

3. Experiments & Results

To test the efficiency of the proposed heuristic, experiments were carried out on 54 standard instances. Such instances were taken from the TSPLIB collection [19] and their vertex set cardinality varies from 100 to 1748 vertices. Non-euclidean instances, such as the ones involving geographical distances (GEO) or special distance functions (ATT), were addressed as well. We recall that the ResNet model was trained on small (100 to 300 vertices) uniform random euclidean instances, evaluated on medium-large (500 to 1000 vertices) uniform random euclidean instances, and tested on TSPLIB instances. We emphasize that TSPLIB instances are generally not uniformly distributed, and, as mentioned, these instances were selected among all available instances in such a way to have no less than 100 total vertices, no more than 2000 total vertices and with the ability to be described in a two-dimensional space.

All the experiments were handled employing python 3.8.5 [54] for the algorithmic component, and pytorch 1.7.1 [55] to manage the neural networks. The following hardware was utilized:

- a single GPU *NVIDIA GeForce GTX 1050 Max-Q*;
- a single CPU *Intel(R) Core(TM) i7-8750H @ 2.20GHz*.

During training, all hardware was exploited, while just the CPU was used to test.

The experiments presented in Table 3 compare *ML-Constructive* (ML-C) results—achieved employing the ResNet—to other famous strategies based on fragments, such as the MF and the CW. The first is equivalent to include all the existing edges in the list of promising edges L_P, sort the list according to the ascending cost values—without considering the CL positions—and then substituting the ML decision taker with a rule that always inserts the considered edge. While the second strategy is equivalent to keeping the list L_P empty, this means that the first phase of *ML-Constructive* does not create any fragment, while the construction is made completely in the second phase. For the sake of completeness, the FI was tested as well and added to Table 3. Even if FI is not a growing fragment approach, and therefore is not an alternative strategy of our *ML-Constructive* heuristic, it is nevertheless a good benchmark solver to compare.

To explore, evaluate and interpret the behavior of our two-phase algorithm, other strategies were investigated as well. The ML decision-taker can act in very different ways, and a comparison with expert-made heuristic rules can be significant. Deterministic and stochastic heuristic rules were created to explore the optimality gap variation. The aim was to prove that the learned ML model would produce a higher gain for the heuristic rules, as corroborated by Table 3. The heuristic rules were substituting the ML decision-taker component within the *ML-Constructive* (lines 10 and 13 of Algorithm 1). No changes in the selecting and sorting strategies were applied to create the lists L_P and L_D. The First (F) rule decides to deterministically add the l edge if one of its extremes is the first closest

vertices in the CL of the other extreme. The Second (S) rule is similar, but it adds l only if one extreme is the second shortest in the CL of the other. The policy that always validates (Y) the insertion of the edges in L_P was examined as well. It represents with CW the extreme cases where the ML decision-taker always validates or not, respectively, the edges in L_P. A stochastic strategy called empirical (E) was tested as well, which adds the edges in L_P according to the distribution seen for the evaluation data-set in Figure 2. Therefore, it inserts the edge if one of the extremes is the first with probability 0.886 or the second with probability 0.512. Twenty runs of the empirical strategy were made, and in (AE) we show the average results, while in (BE) we show the best from all runs. Finally, to check the behavior of the *ML-Constructive* in case the ML system validates with 100% accuracy (the ML decision-taker is a perfect oracle), the Super Confident (ML-SC) policy was examined. This policy always answers correctly for all the edges in the promising list L_P, and is achieved by exploiting the known optimal tours. To capture the potentiality of a super-accurate network for the first phase, the partial solution created by the ML-SC policy in the first phase and the solution constructed in the second phase are shown in Figure 7. Note that some crossing edges are created in the second phase. In fact, despite the solution created being very close to the optimal, the second phase sometimes adds bad edges to the solution. This last artificial policy has been added to demonstrate how much leverage we can still gain from the ML point of view.

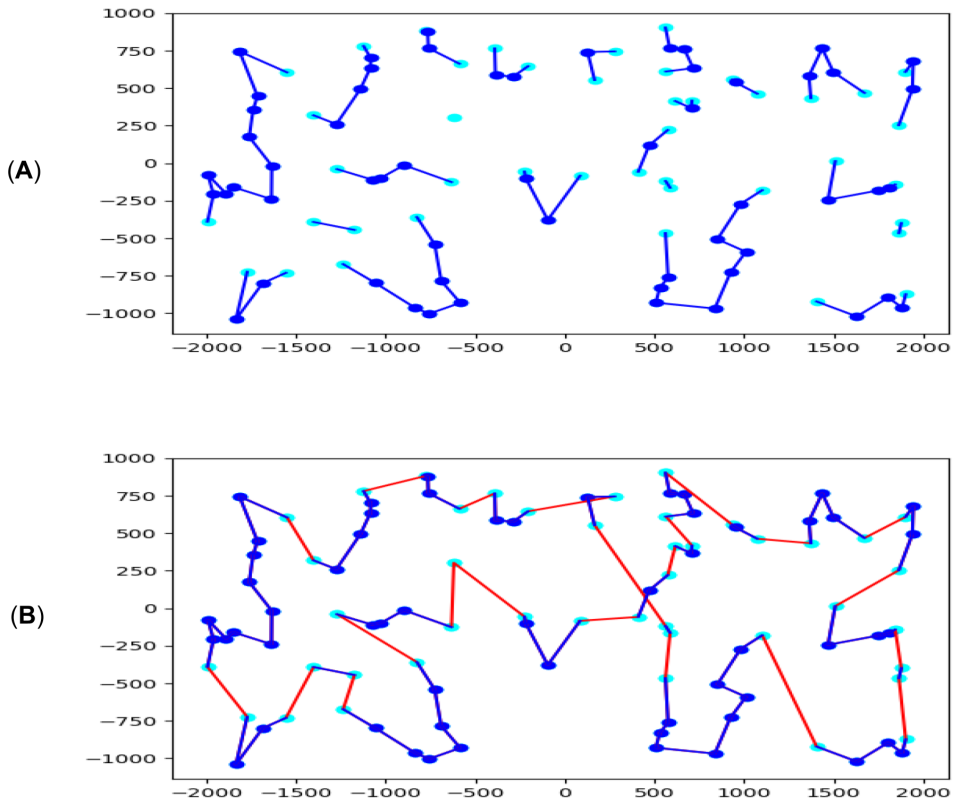

Figure 7. (**A**) The partial solution available at the end of the first phase for ML-SC. In light blue the remaining free vertices, and in dark blue the inserted edges. (**B**) The complete tour was found at the end of the ML-SC run. In red the edges were added during the second phase. The considered instance is the KroA100 from the TSPLIB collection.

The results obtained by the optimal policy (ML-SC) lead us to two interesting aspects. The first one, as mentioned before, shows the possible leverage from the ML perspective (first phase). While the second one gives us an idea of how much improvement is possible from the heuristic point of view (second phase). To emphasize these aspects the ML-SC (gap) column in Table 3 shows the difference, in terms of percentage error, between the

ML-SC solution and the best solution found by the other heuristics (in bold). In fact, the top nine columns were compared with each other to find the best solution, while the tenth column (ML-SC) was later compared with all the others. The average, the standard deviation (std), and the number of times when the heuristic is best are shown as well for each strategy. The gap column is of interest since it reveals that occasionally the solution is highly affected by the behavior of the second phase of the heuristic. Even if the heuristic behaves greatly for promising edges the second phase can still ruin the solution.

Among the many policies shown in Table 3, the First (F) and the ML-based (ML-C) policies exhibit comparable average gaps. To prove that the enhancement introduced by the ML system is on average statistically significant, a statistical test was conducted. A T-test on the percentage errors obtained for the 54 instances in Table 3 shows that the p-value against the hypothesis both policies are similar in terms of the average optimal gap is equal to 0.03. The result proves that the enhancement is relevant, and that these systems have a promising role in improving the quality of TSP solvers.

To check the behavior of the heuristics concerning time, Table 4 shows the CPU time for each policy and heuristic shown in Table 3. Note that for each query to the ResNet the input procedure produces an image that increases the computational burden. Therefore, future work could be proposed to speed the ML component, even though the computation times remain short and acceptable for many online optimization scenarios.

Finally, to make a comparison with the metrics presented for the MF, FI, and CW in Section 2.2, the results in Table 5 show the final tour achievements for the F, ML-C, and ML-SC policies across the various positions in the CL. Note that although the TPR of ML-SC is 100%, its FPR is not equal to zero since during the second phase some edges in the first and second position can be inserted. Also note that the accuracy of ML-C is consistently better than F, while the FPR for the first position is lower resulting in a higher TPR for the second position.

Table 3. Percentage error comparison of various decision-takers policies for testing TSPLIB instances.

Instances	MF	FI	CW	F	S	Y	AE	BE	ML-C	ML-SC	ML-SC (Gap)
kroA100	14.120	16.596	**6.043**	9.618	22.437	8.636	11.986	7.429	6.480	**3.792**	2.251
kroC100	12.270	**4.979**	11.480	8.362	27.558	5.263	13.391	6.950	10.343	**4.776**	0.203
rd100	16.928	11.214	8.736	11.580	24.121	11.214	14.212	8.938	**8.559**	6.738	1.821
eil101	27.504	12.719	5.087	8.426	22.099	18.760	14.348	8.426	**4.293**	**0.000**	4.293
lin105	16.065	9.006	8.638	7.177	37.332	21.406	12.580	**7.080**	8.485	10.780	-3.700
pr107	5.799	**2.742**	10.166	9.245	10.640	8.936	9.454	6.717	11.153	**0.445**	2.297
pr124	10.110	8.225	**2.502**	4.911	24.344	7.942	9.079	4.191	6.991	**2.997**	-0.495
bier127	14.186	16.151	5.659	4.753	25.162	12.370	10.091	**4.571**	6.604	**0.000**	4.571
ch130	28.462	13.912	7.480	7.414	22.733	8.003	12.305	8.429	**4.975**	**4.206**	0.769
pr136	23.160	10.089	**7.186**	11.709	15.693	16.046	14.878	11.701	11.151	**3.713**	3.473
gr137	27.234	13.502	8.243	9.742	30.207	15.548	14.170	10.124	**7.329**	**3.188**	4.141
pr144	12.483	3.965	6.444	6.628	12.016	4.161	7.625	**3.796**	4.474	**3.962**	-0.166
kroA150	20.238	15.876	8.468	10.507	26.934	14.134	12.799	9.467	**6.877**	**1.139**	5.738
pr152	15.196	7.022	9.455	7.204	17.060	6.117	8.239	**5.647**	6.919	**3.793**	1.854
u159	17.952	25.604	8.408	9.009	19.881	10.542	11.788	**5.589**	7.952	**6.024**	-0.435
rat195	13.043	15.497	**5.854**	7.576	13.431	13.345	10.286	**5.854**	7.533	**0.000**	5.854
d198	20.507	8.251	**5.444**	6.711	17.535	7.744	9.262	6.267	6.255	**4.011**	1.433
kroA200	17.819	12.732	8.622	11.097	25.541	11.298	13.008	10.328	**6.681**	**2.094**	4.587
gr202	15.935	10.323	5.683	7.367	19.916	7.716	9.673	6.331	**4.436**	**1.825**	2.611
ts225	12.842	24.528	6.804	9.493	6.975	14.929	11.309	8.075	**6.520**	11.330	-4.810
tsp225	26.237	15.644	10.438	9.790	24.165	9.609	13.906	**9.169**	11.292	**4.403**	4.766
pr226	21.052	**1.788**	9.948	11.918	16.784	9.031	12.347	8.778	8.599	**5.370**	-3.582
gr229	19.624	24.001	8.849	7.593	22.242	10.932	10.807	**6.573**	7.495	**2.807**	3.766
gil262	12.279	20.732	9.714	6.602	24.769	10.892	10.976	8.368	**6.224**	8.999	-2.775
pr264	14.987	13.259	8.839	**4.919**	16.239	9.816	10.091	6.195	6.036	**3.739**	1.180
a280	20.822	14.153	13.998	13.959	15.394	12.447	15.944	12.330	**11.439**	**0.388**	11.051
pr299	21.639	15.903	**8.103**	10.467	22.371	15.244	14.385	10.965	8.636	**0.905**	7.198
lin318	18.356	20.750	7.849	**6.377**	30.848	14.114	12.652	9.398	6.679	**5.501**	0.876
rd400	15.032	18.265	9.548	8.278	21.923	10.844	11.840	**7.506**	8.108	**3.423**	4.083
fl417	12.469	15.589	12.335	10.606	28.488	8.962	12.149	8.473	**8.102**	**7.394**	0.708

Table 3. *Cont.*

Instances	MF	FI	CW	F	S	Y	AE	BE	ML-C	ML-SC	ML-SC (Gap)
gr431	19.672	16.603	11.953	**6.942**	21.114	11.550	12.270	7.867	11.554	**4.196**	2.746
pr439	15.983	15.487	14.609	9.024	23.183	13.395	12.157	9.439	**7.661**	**7.104**	0.557
pcb442	21.423	17.988	**9.935**	12.460	16.403	13.908	13.233	11.596	10.172	**2.787**	7.148
d493	16.550	17.172	8.652	9.618	17.898	9.872	9.749	8.443	**6.818**	**4.352**	2.466
att532	22.223	18.256	11.013	**7.596**	25.022	11.150	11.629	9.455	8.441	**3.590**	4.006
u574	22.276	23.818	10.779	**9.023**	24.349	9.638	13.113	9.776	9.790	**4.495**	4.528
rat575	18.529	24.376	8.460	10.350	19.592	8.962	11.734	9.671	**6.201**	**2.761**	3.440
d657	13.997	19.781	7.949	**7.583**	23.008	8.878	12.009	10.043	7.587	**3.829**	3.754
gr666	13.473	19.277	13.241	**9.314**	24.339	12.670	13.736	11.534	9.635	**6.741**	2.573
u724	17.836	23.298	9.881	7.590	23.028	9.952	11.276	9.308	**6.543**	**3.054**	3.489
rat783	22.062	22.333	9.642	7.047	21.549	11.617	10.995	8.877	**5.934**	**4.956**	0.978
pr1002	18.857	24.818	10.763	9.751	20.484	13.648	13.238	11.600	**8.529**	**5.364**	3.165
u1060	17.322	23.213	10.732	9.620	21.754	10.537	13.128	11.580	**8.954**	**6.537**	2.417
vm1084	23.083	22.962	10.298	9.615	28.746	12.485	13.390	11.253	**9.123**	**6.951**	2.172
pcb1173	17.792	26.408	10.917	**9.567**	21.033	14.631	13.380	11.821	9.986	**5.790**	3.777
d1291	21.917	22.502	10.155	**5.711**	14.783	9.653	9.643	7.573	8.535	**3.081**	2.630
rl1304	12.142	28.188	10.610	**7.512**	25.060	11.100	11.821	9.580	9.506	**5.228**	2.284
rl1323	14.876	26.432	11.804	7.467	25.401	8.385	11.310	9.789	**6.538**	**3.695**	2.843
nrw1379	22.314	25.211	9.914	8.842	19.794	11.351	11.443	9.856	**7.996**	**3.655**	4.341
fl1400	20.520	13.733	11.432	10.975	27.267	13.718	14.541	**9.997**	11.273	**8.471**	1.526
u1432	23.329	21.065	10.407	12.676	14.967	14.928	14.669	12.491	**8.440**	**5.725**	2.715
fl1577	16.976	23.053	12.167	12.634	14.082	12.351	12.502	**9.084**	10.463	**4.373**	4.711
d1655	15.282	20.785	11.187	9.503	18.427	10.368	12.130	9.478	**8.269**	**5.637**	2.632
vm1748	14.134	24.335	11.912	9.992	24.520	11.868	13.757	12.225	**9.302**	**6.094**	3.208
average	17.906	17.113	9.341	8.879	21.493	11.345	12.082	8.815	**8.035**	**4.374**	2.549
std	4.659	6.652	2.368	2.077	5.493	3.142	1.789	2.155	**1.875**	**2.473**	2.715
best	0/54	3/54	6/54	10/54	0/54	0/54	0/54	11/54	24/54	50/54	

Table 4. CPU time comparison related to different greedy policies for testing TSPLIB instances.

Instances	MF	FI	CW	F	S	Y	AE	BE	ML-C	ML-SC
kroA100	**0.004**	0.006	0.006	0.009	0.008	0.007	0.014	0.276	1.423	0.011
kroC100	**0.004**	0.006	0.006	0.011	0.010	0.008	0.015	0.309	1.706	0.010
rd100	**0.004**	0.006	0.008	0.011	0.010	0.008	0.016	0.323	1.608	0.011
eil101	**0.006**	0.006	0.007	0.012	0.011	0.011	0.019	0.371	2.178	0.013
lin105	**0.005**	0.006	0.007	0.013	0.012	0.011	0.017	0.335	1.445	0.012
pr107	**0.005**	0.007	0.009	0.013	0.011	0.011	0.015	0.302	1.298	0.012
pr124	**0.005**	0.009	0.009	0.017	0.017	0.016	0.020	0.395	1.565	0.015
bier127	0.010	**0.009**	0.012	0.100	0.097	0.047	0.055	1.106	2.554	0.059
ch130	0.011	**0.009**	0.011	0.017	0.016	0.015	0.024	0.473	2.507	0.019
pr136	**0.009**	0.020	0.011	0.015	0.014	0.014	0.022	0.438	2.308	0.017
gr137	**0.006**	0.011	0.012	0.023	0.021	0.020	0.027	0.547	2.557	0.024
pr144	**0.006**	0.012	0.015	0.023	0.022	0.018	0.024	0.487	0.640	0.021
kroA150	**0.011**	0.014	0.012	0.021	0.017	0.017	0.025	0.492	2.444	0.019
pr152	**0.010**	0.013	0.015	0.023	0.021	0.019	0.026	0.528	1.047	0.022
u159	**0.007**	0.014	0.015	0.026	0.025	0.024	0.029	0.580	2.926	0.026
rat195	**0.009**	0.021	0.024	0.027	0.027	0.026	0.038	0.765	3.553	0.025
d198	0.022	**0.020**	0.027	0.111	0.109	0.106	0.115	2.305	3.451	0.106
kroA200	**0.011**	0.021	0.028	0.036	0.032	0.024	0.044	0.879	3.975	0.027
gr202	0.024	**0.022**	0.028	0.137	0.136	0.129	0.143	2.866	4.018	0.125
ts225	**0.012**	0.027	0.036	0.043	0.044	0.034	0.044	0.884	3.955	0.036
tsp225	**0.019**	0.028	0.034	0.041	0.038	0.033	0.052	1.036	4.239	0.031
pr226	**0.018**	0.031	0.031	0.055	0.052	0.051	0.056	1.111	1.102	0.054
gr229	**0.019**	0.031	0.032	0.091	0.083	0.088	0.100	1.996	4.220	0.080
gil262	**0.017**	0.035	0.043	0.058	0.061	0.042	0.068	1.354	4.640	0.060
pr264	**0.030**	0.037	0.037	0.066	0.067	0.053	0.070	1.401	3.815	0.055
a280	**0.032**	0.043	0.044	0.524	0.502	0.473	0.503	10.064	5.368	0.471
pr299	**0.037**	0.050	0.062	0.073	0.060	0.059	0.088	1.770	5.429	0.064
lin318	**0.026**	0.057	0.069	0.091	0.084	0.079	0.099	1.977	4.871	0.090

Table 4. *Cont.*

Instances	MF	FI	CW	F	S	Y	AE	BE	ML-C	ML-SC
rd400	**0.042**	0.095	0.096	0.127	0.112	0.119	0.151	3.013	7.594	0.129
fl417	**0.048**	0.114	0.108	0.211	0.190	0.192	0.221	4.425	7.020	0.193
gr431	**0.076**	0.110	0.133	0.359	0.353	0.352	0.396	7.927	8.016	0.342
pr439	**0.094**	0.123	0.130	0.231	0.199	0.194	0.241	4.817	7.093	0.197
pcb442	**0.088**	0.133	0.147	0.193	0.179	0.151	0.203	4.055	8.526	0.157
d493	**0.144**	0.231	0.210	0.884	0.898	0.886	0.907	18.141	10.235	0.889
att532	**0.111**	0.201	0.219	0.309	0.265	0.290	0.329	6.571	10.010	0.230
u574	**0.154**	0.219	0.218	0.297	0.272	0.281	0.339	6.789	10.652	0.291
rat575	**0.105**	0.233	0.225	0.236	0.223	0.186	0.282	5.647	11.079	0.204
d657	**0.159**	0.332	0.294	1.025	1.034	1.003	1.081	21.617	12.054	1.028
gr666	**0.170**	0.651	0.381	0.768	0.651	0.706	0.808	16.150	12.513	0.680
u724	**0.149**	0.455	0.394	0.338	0.359	0.310	0.465	9.298	12.734	0.290
rat783	0.303	0.504	0.448	0.383	0.389	0.314	0.532	10.647	14.551	**0.299**
pr1002	**0.498**	1.042	0.989	0.879	0.758	0.766	1.099	21.974	20.139	0.747
u1060	**0.294**	1.183	0.802	1.401	1.198	1.057	1.444	28.873	20.074	1.149
vm1084	**0.412**	1.504	0.887	1.061	0.867	0.712	1.152	23.045	17.057	0.722
pcb1173	**0.402**	1.657	0.956	1.109	1.050	0.997	1.458	29.156	22.952	0.947
d1291	**0.918**	2.192	1.247	4.518	4.452	4.138	4.289	85.775	20.517	3.881
rl1304	**0.538**	2.482	1.303	1.284	1.145	1.069	1.574	31.470	19.792	1.075
rl1323	**0.696**	1.856	1.332	1.739	1.655	1.268	1.861	37.213	20.215	1.324
nrw1379	**0.994**	1.942	1.624	1.270	1.203	1.107	1.762	35.249	29.439	1.113
fl1400	**0.745**	2.020	1.228	2.655	2.674	2.559	2.816	56.325	30.401	2.583
u1432	**1.050**	2.116	1.917	1.954	1.760	1.422	2.104	42.074	32.086	1.072
fl1577	**1.044**	2.733	2.117	2.650	2.616	1.840	2.713	54.267	25.673	1.438
d1655	**1.551**	3.229	2.689	8.131	7.945	7.751	8.623	172.450	30.257	7.624
vm1748	**0.660**	3.914	2.360	2.525	1.836	1.914	2.894	57.884	28.875	1.967
average	**0.219**	0.590	0.428	0.708	0.665	0.612	0.769	15.374	9.822	0.594
std	**0.351**	0.971	0.679	1.364	1.319	1.251	1.433	28.660	9.280	1.217

Table 5. TPR, FPR, accuracy and precision comparison across several positions and policies.

Position	Method	TPR	FPR	Accuracy	Precision
1	F	98.07%	85.80%	86.68%	87.91%
	ML-C	93.64%	53.08%	87.29%	91.82%
	ML-SC	100.00%	3.41%	99.54%	99.47%
2	F	68.28%	18.12%	73.62%	85.35%
	ML-C	84.74%	29.07%	79.32%	81.84%
	ML-SC	100.00%	3.01%	98.82%	98.09%
3	F	44.40%	8.28%	80.39%	62.82%
	ML-C	42.74%	6.02%	81.71%	69.11%
	ML-SC	86.27%	0.91%	96.02%	96.74%
4	F	38.09%	5.75%	87.26%	48.47%
	ML-C	33.08%	3.60%	88.52%	56.62%
	ML-SC	80.27%	0.94%	96.73%	92.41%
5	F	31.18%	4.42%	91.95%	29.66%
	ML-C	26.52%	1.94%	94.02%	44.92%
	ML-SC	73.95%	0.88%	97.70%	83.42%
>5	F	28.94%	0.02%	99.97%	13.62%
	ML-C	25.76%	0.02%	99.97%	11.83%
	ML-SC	64.24%	0.01%	99.99%	48.84%

4. Discussion

A new strategy to design constructive heuristics has been presented. It gives a central role to the integration of statistical, mathematical, and heuristic exploration. We introduced a new way of thinking about the generalization of ML approaches for the TSP, leading to an efficient integration between learning useful information and exploiting it through classic approaches. The objective is to learn useful skills from experience to enhance the

heuristic search. Our *ML-Constructive* is the first ML approach able to scale and show improvements at once with respect to a classic efficient constructive heuristic. Furthermore, the introduced approach can give good guidelines about how the ML can behave in the event of extreme negative or positive cases. Results are very promising and suggest that giving more emphasis on the generalization of hybrid designs pays off.

The relevance of an exploratory stage with statistical studies of the problem at hand had been emphasized. The target of these studies is to select an effective sub-problem that allows the avoidance of many known ML flaws.

More work needs to be done to improve the accuracy and the extrapolation of the ML classifier. Further improvements in future work could be in the direction of reducing the (constant) time required to prepare the input for the ML classifier, and to find the integration to meta-heuristics approaches as well.

Author Contributions: Conceptualization, U.J.M., L.M.G. and R.M.; methodology, U.J.M.; software, U.J.M.; validation, U.J.M., L.M.G. and R.M.; formal analysis, U.J.M.; investigation, U.J.M.; resources, U.J.M.; data curation, U.J.M.; writing—original draft preparation, U.J.M.; writing—review and editing, U.J.M. and R.M.; visualization, U.J.M.; supervision, R.M.; project administration, R.M.; funding acquisition, R.M. All authors have read and agreed to the published version of the manuscript.

Funding: Umberto Junior Mele was supported by the Swiss National Science Foundation through grants 200020-182360: "Machine learning and sampling-based metaheuristics for stochastic vehicle routing problems".

Data Availability Statement: All the code for the experiments replication and for the data-sets creation can be found in the github repository: https://github.com/UmbertoJr/ML-Constructive (accessed on 8 September 2021).

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

Appendix A. Complexity of the Inner-Loop Constraint Tracker

The purpose here is to compute the complexity of the inner-loop constraint tracker used in the *ML-Constructive* heuristic process, as stated in Section 2.4 and in Algorithm 1. For comprehension purposes we strict our analysis to the symmetric TSP, but similar results can be achieved for the asymmetric case as well.

Firstly, we observe that the constraint tracker procedure is applied only to edges that have both extremes with exactly one connection already in the partial solution, since the tracker routine follows the constraints expressed by lines 8 and 22 in Algorithm 1. Therefore, edges connecting vertices from internal points of the fragments are impossible to occur at this point, as shown in Figure A1. While those creating an inner-loop and joining two fragments are possible to occur events (Figure A2). Note that the goal of the tracker is to detect the inner-loop connections from the other. The growing connections and new fragment connections shown in Figure A2 are events that can be detected in constant time, since it's enough to check that an extreme of the inserting edge has zero connections in the partial solution (lines 12 and 25). Also, note that these two events cannot occur as input of the tracker procedure since they do not satisfy the constraint expressed by lines 8 and 22.

Secondly, we notice that the complexity of the worst-case scenario for the whole procedure (from empty solution to the complete) is being computed in this Appendix. Therefore, we are not taking into consideration just the single call of the tracker, but the global computation during the complete tracking process. In fact, considering that the maximum number of positive addition for a constructive heuristic that grows fragments is equal to n (the length of the tour). Where, an addition is positive if the edge being attached to the partial solution complies with the TSP constraints in (1b-e) and the ML decision-taker agrees to add the considered edge in solution. We refer to the epoch between two positive additions as t, e.g., no edge is in solution at $t = 0$, meanwhile exactly eight edges are in solution at $t = 8$. Take into consideration that the epochs to be checked by the tracking routine for the symmetric TSP are from $t = 2$ to $t = n - 2$.

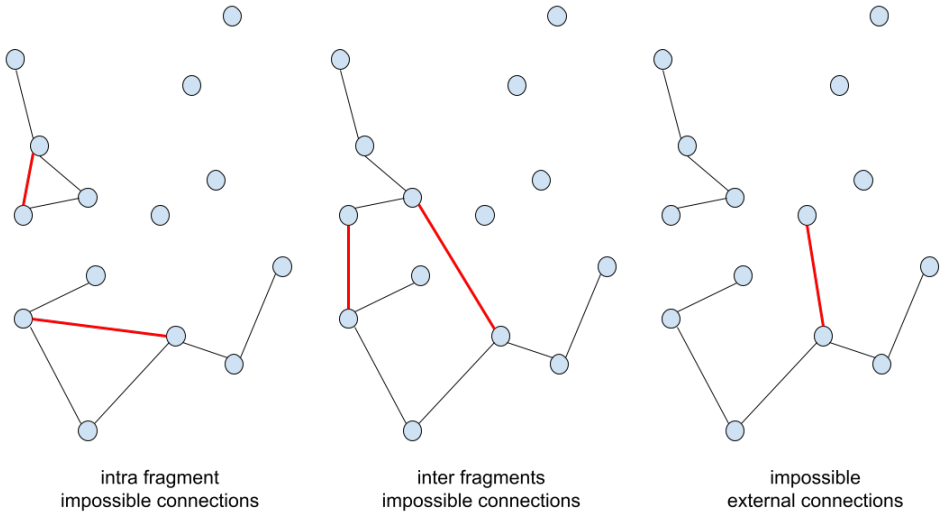

Figure A1. Events that cannot occur as an input to the tracking procedure.

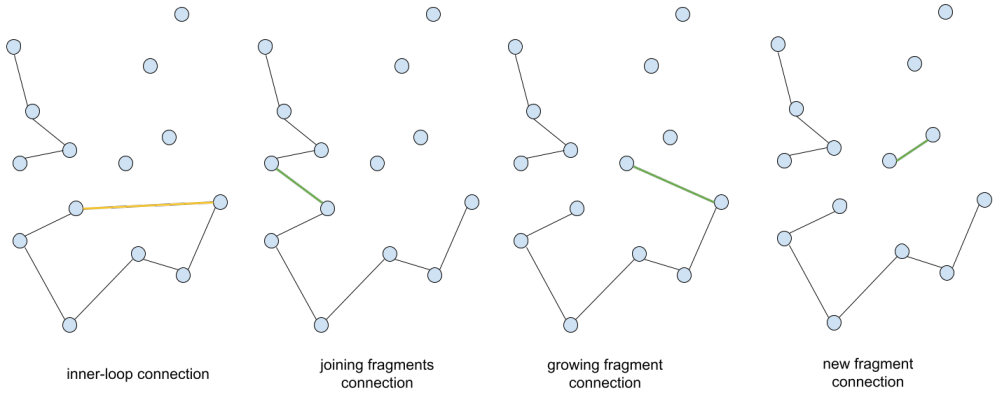

Figure A2. Events that can occur and are prevented by the tracking procedure (first two), and events that can be detected in constant time (last two).

As mentioned, the computationally expensive events that the tracker needs to check are the "inner-loop connection" and the "joining fragments connections". The inner-loop connection drawn in yellow (Figure A2) occurs when the extremes of a fragment are connected together by the attaching edge l. If we assume that at the epoch t there are at most $s \leq t$ fragments, then there exist at most s attaching edges at this epoch that can create an inner loop (Figure A3), and the sum of the operation needed to check these s inner-loops is at most equal to t. In fact, the tracker checks by spanning completely one of the fragments connecting to the attaching edge. Then if the other extreme of the fragment coincides with the other extreme of the attaching edge there is an "inner loop connection", otherwise there is a "joining fragments connection". Note that once an edge has been rejected the fragment associated with it is set free for the current epoch, and the tracker does not need to check anymore its extremes. Since we have at most t operations for epoch, and we have at most n epochs, the global computation is $O(n^2)$.

Once the upper bound of the complexity for detecting the "inner-loop connections" has been found, the number of operations required for the "joining fragments connections" occurrences is still necessary to be estimated. Usually, after having encountered a "joining fragments connection" event, the insertion of the considered edge l takes place. But since in *ML-Constructive* it could happen that the ML decision-taker rejects the attaching edge (line 10), it may happen that the tracker is called many times during the same epoch. This could be a problem if the promising list L_P was not limited at most $m \times n$ edges

(Section 2.3). Assuming that the worst-case scenario is $O(n)$ for each edge processed in the first phase, we are still safe with $O(n^2)$ operations for the global tracking computation.

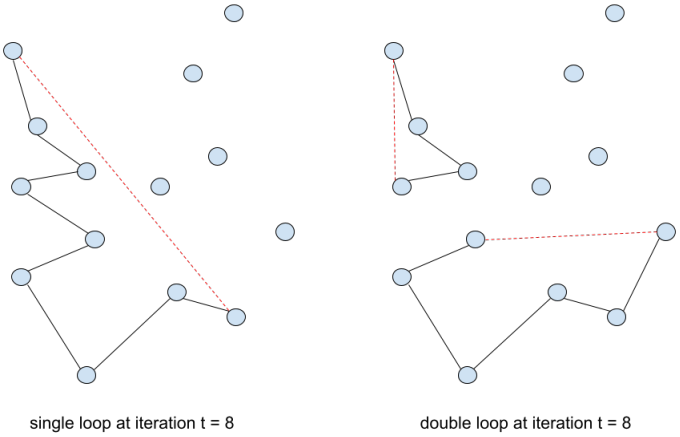

Figure A3. Single and double fragments possible inner-loops at the eighth epoch.

Appendix B. The Earlier Insertion of the Most Promising Edges Could Increase the Probability of Finding the Optimal Tour

The purpose of this Appendix is to present some advantages that a procedure that inserts promising edges into a solution first has with respect to other approaches. If the growing fragments heuristic is considered as a stochastic process, then we could estimate the probability that the optimal tour has of occurring following the procedure. In fact, for each edge l considered to be included in the partial solution there are two possible events: included or not. If the random variable E_l is used to refer to the event that the edge l is included in the solution ($\neg E_l$ otherwise), then we can express the probability that such event occurs as:

$$P(E_l) = 1 - P(A_l) - P(B_l) \quad \text{and} \quad P(\neg E_l) = P(A_l) + P(B_l) \tag{A1}$$

where A_l refers to the internal point connection events (Figure A1), while B_l stand for the inner-loop connection events (Figure A2). Recall that internal point connection occurs when the constraint which ensures that no vertex is connected to more than two other vertices is not satisfied. While an inner-loop occurs when sub-solutions are generated, instead of having a single global loop.

In case that a list L is used to store all the existing edges of the TSP instance that we wish to solve, and we randomly shuffle such a list to create a random examination order. The probabilities of the events A_l and B_l will be dependent on the position p in such a list and the number of edges already inserted in the partial solution. So, combinatorics can help us calculate or approximate these probabilities. Recalling the epoch concept described in Appendix A, we can state that at $t = 0$ such probability is one, while at $t = n$ the probability of E_l is zero:

$$P(E_l | \, t = 0) = 1 \tag{A2}$$

$$P(E_l | \, t = n) = 0 \tag{A3}$$

Since at $t = 0$ no edge has been placed in solution, no A_l or B_l event can occur. While at $t = n$ the solution is complete. Then, we want to prove that the probability of E_l will monotonically decrease as more edges are fed into the solution and as we progress through the L list. In case this conjecture is true, we can conclude that edges inspected earlier in the list are more likely to be included than those seen later. Emphasising the need to put first in the L list the edges that we consider most promising to be included in the optimal solution. As a first step, we determine the probability of occurrence of A_l. To figure out

such probability, we shall simply estimate the number of cases in which A_l occurs and divide by the total number of possible cases. These cases vary depending on the position p in the list, the number of edges e, and the number of vertices n in the instance. Recalling that A_l occurs when in the list the edge l is preceded by at least 2 other edges that have the same extreme with that of l. In case there are d of these overlapping edges, we have that A_l occur for $d = 2$ to $d = n - 1$:

$$P(A_l|\, p) = \frac{\sum_{d=2}^{n-1} \binom{n-1}{d}\binom{e-n+1}{p-1-d}}{\binom{e}{p-1}} \approx \sum_{d=2}^{n-1} \frac{(p-1)!}{(p-1-d)!} \tag{A4}$$

which is an increasing function with respect to the position p, and converge to 1 as p goes to e.

Meanwhile, to compute the probability of B_l, the epoch in which the event occurs must be taken into account. Bearing in mind that as we proceed along with the p positions in the list such epoch is ascending, since there are no operations that remove an edge from the solution and it is possible just to add a new edge into such a partial solution.

Considering that the maximum t total number of inner-loops for a given epoch is fixed and equal to t, we have that:

$$P(B_l|\, p,\, t) < \frac{t}{e-p-1} \qquad \text{with} \quad t \leq n \tag{A5}$$

which has an upper bound that is an increasing function with respect to the position p and the epoch t.

To conclude, since the probabilities of A_l and B_l show an increasing trend, although not strictly due to the upper bound of B_l, we can verify that the probability of E_l has a decreasing trend due to Equation (A1). Therefore, the anticipation of the insertion of promising edges is a good strategy for the heuristic. However, such results do not prove that for any solving algorithm, the probability $P(E_L|\, t)$ is a strictly decreasing function. But it suggests that a general decreasing trend is present which should be exploited by the *ML-Constructive* heuristic during the construction of the solution.

References

1. Applegate, D.L.; Bixby, R.E.; Chvátal, V.; Cook, W.J. *The Traveling Salesman Problem: A Computational Study*; Princeton Series in Applied Mathematics; Princeton University Press: Princeton, NJ, USA, 2006.
2. Matai, R.; Singh, S.P.; Mittal, M.L. Traveling Salesman Problem: An Overview of Applications, Formulations, and Solution Approaches. Available online: https://www.intechopen.com/chapters/12736 (accessed on 8 September 2021).
3. Held, M.; Karp, R.M. A dynamic programming approach to sequencing problems. *J. Soc. Ind. Appl. Math.* **1962**, *10*, 196–210. [CrossRef]
4. Dorigo, M.; Gambardella, L.M. Ant colonies for the travelling salesman problem. *Biosystems* **1997**, *43*, 73–81. [CrossRef]
5. Helsgaun, K. An effective implementation of the Lin–Kernighan traveling salesman heuristic. *Eur. J. Oper. Res.* **2000**, *126*, 106–130. [CrossRef]
6. Dell'Amico, M.; Montemanni, R.; Novellani, S. Modeling the flying sidekick traveling salesman problem with multiple drones. *Networks* **2021**. [CrossRef]
7. Caserta, M.; Voß, S. A hybrid algorithm for the DNA sequencing problem. *Discret. Appl. Math.* **2014**, *163*, 87–99. [CrossRef]
8. Montemanni, R.; Gambardella, L.M. Minimum power symmetric connectivity problem in wireless networks: A new approach. In *Mobile and Wireless Communication Networks, Proceedings of the IFIP International Conference on Mobile and Wireless Communication Networks, Paris, France, 25–27 October 2004*; Springer: Boston, MA, USA, 2004; pp. 497–508.
9. MacGregor, J.N.; Ormerod, T. Human performance on the traveling salesman problem. *Percept. Psychophys.* **1996**, *58*, 527–539. [CrossRef] [PubMed]
10. Mele, U.J.; Gambardella, L.M.; Montemanni, R. Machine Learning Approaches for the Traveling Salesman Problem: A Survey. In Proceedings of the IEEE 8th International Conference on Industrial Engineering and Applications, Kyoto, Japan, 23–26 April 2021; Association for Computing Machinery: New York, NY, USA, 2021.
11. Bengio, Y.; Lodi, A.; Prouvost, A. Machine learning for combinatorial optimization: A methodological tour d'horizon. *Eur. J. Oper. Res.* **2020**, *290*, 405–421. [CrossRef]
12. Kool, W.; Van Hoof, H.; Welling, M. Attention, learn to solve routing problems! *arXiv* **2018**, arXiv:1803.08475.

13. Mele, U.J.; Chou, X.; Gambardella, L.M.; Montemanni, R. Reinforcement Learning and Additional Rewards for the Traveling Salesman Problem. In Proceedings of the IEEE 7th International Conference on Industrial Engineering and Applications, Bangkok, Thailand, 16–18 April 2020; Association for Computing Machinery: New York, NY, USA, 2021.
14. da Costa, P.R.; Rhuggenaath, J.; Zhang, Y.; Akcay, A. Learning 2-opt Heuristics for the Traveling Salesman Problem via Deep Reinforcement Learning. In Proceedings of the 12th Asian Conference on Machine Learning, Bangkok, Thailand, 18–20 November 2020; PMLR: New York, NY, USA, 2020; pp. 465–480.
15. Zheng, J.; He, K.; Zhou, J.; Jin, Y.; Li, C.M. Combining reinforcement learning with lin-kernighan-helsgaun algorithm for the traveling salesman problem. *arXiv* **2020**, arXiv:2012.04461.
16. Fu, Z.H.; Qiu, K.B.; Zha, H. Generalize a Small Pre-trained Model to Arbitrarily Large TSP Instances. *arXiv* **2020**, arXiv:2012.10658.
17. Zohuri, B.; Moghaddam, M. Deep learning limitations and flaws. *Mod. Approaches Mater. Sci.* **2020**, *2*, 241–250.
18. Marcus, G. Deep learning: A critical appraisal. *arXiv* **2018**, arXiv:1801.00631.
19. Reinelt, G. TSPLIB—A traveling salesman problem library. *Orsa J. Comput.* **1991**, *3*, 376–384. [CrossRef]
20. He, K.; Zhang, X.; Ren, S.; Sun, J. Deep residual learning for image recognition. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, Las Vegas, NV, USA, 27–30 June 2016; pp. 770–778.
21. Dantzig, G.; Fulkerson, R.; Johnson, S. Solution of a large-scale traveling-salesman problem. *J. Oper. Res. Soc. Am.* **1954**, *2*, 393–410. [CrossRef]
22. Steiglitz, K. Some improved algorithms for computer solution of the traveling salesman problem. In *Proceedings of the 6th Annual Allerton Conference on Communication, Control, and Computing*; Department of Electrical Engineering and the Coordinated Science Laboratory, University of Illinois: Champaign, IL, USA, 1968. Available online: https://www.researchgate.net/publication/201976955_Some_Improved_Algorithms_for_Computer_Solution_of_the_Traveling_Salesman_Problem (accessed on 8 September 2021).
23. Bentley, J.L. Experiments on traveling salesman heuristics. In Proceedings of the First Annual ACM-SIAM Symposium on Discrete Algorithms, San Francisco, CA, USA, 22–24 January 1990; pp. 91–99.
24. Clarke, G.; Wright, J.W. Scheduling of vehicles from a central depot to a number of delivery points. *Oper. Res.* **1964**, *12*, 568–581. [CrossRef]
25. Johnson, D.S.; McGeoch, L.A. Experimental Analysis of Heuristics for the STSP. In *The Traveling Salesman Problem and Its Variations*; Springer: Boston, MA, USA, 2007; pp. 369–443.
26. Reinelt, G. *The Traveling Salesman: Computational Solutions for TSP Applications*; Springer: Berlin/Heidelberg, Germany, 2003; Volume 840
27. Vinyals, O.; Fortunato, M.; Jaitly, N. Pointer networks. *arXiv* **2015**, arXiv:1506.03134.
28. Deudon, M.; Cournut, P.; Lacoste, A.; Adulyasak, Y.; Rousseau, L.M. Learning heuristics for the tsp by policy gradient. In *Integration of Constraint Programming, Artificial Intelligence, and Operations Research*; Springer: Cham, Switzerland, 2018; pp. 170–181.
29. Dai, H.; Khalil, E.B.; Zhang, Y.; Dilkina, B.; Song, L. Learning combinatorial optimization algorithms over graphs. *arXiv* **2017**, arXiv:1704.01665.
30. Ma, Q.; Ge, S.; He, D.; Thaker, D.; Drori, I. Combinatorial optimization by graph pointer networks and hierarchical reinforcement learning. *arXiv* **2019**, arXiv:1911.04936.
31. Taillard, É.D.; Helsgaun, K. POPMUSIC for the travelling salesman problem. *Eur. J. Oper. Res.* **2019**, *272*, 420–429. [CrossRef]
32. Lee, D.T.; Schachter, B.J. Two algorithms for constructing a Delaunay triangulation. *Int. J. Comput. Inf. Sci.* **1980**, *9*, 219–242. [CrossRef]
33. Fitzpatrick, J.; Ajwani, D.; Carroll, P. Learning to Sparsify Travelling Salesman Problem Instances. *arXiv* **2021**, arXiv:2104.09345.
34. Bello, I.; Pham, H.; Le, Q.V.; Norouzi, M.; Bengio, S. Neural combinatorial optimization with reinforcement learning. *arXiv* **2016**, arXiv:1611.09940.
35. Mazyavkina, N.; Sviridov, S.; Ivanov, S.; Burnaev, E. Reinforcement learning for combinatorial optimization: A survey. *Comput. Oper. Res.* **2021**, *134*, 105400. [CrossRef]
36. Konda, V.R.; Tsitsiklis, J.N. Actor-critic algorithms. In *Advances in Neural Information Processing Systems*; Citeseer: Princeton, NJ, USA, 2000; pp. 1008–1014.
37. Christofides, N. *Worst-Case Analysis of a New Heuristic for the Travelling Salesman Problem*; Technical Report; Carnegie-Mellon Univ Pittsburgh Pa Management Sciences Research Group: Pittsburgh, PA, USA, 1976.
38. Joshi, C.K.; Cappart, Q.; Rousseau, L.M.; Laurent, T.; Bresson, X. Learning TSP requires rethinking generalization. *arXiv* **2020**, arXiv:2006.07054.
39. Miki, S.; Ebara, H. Solving Traveling Salesman Problem with Image-Based Classification. In Proceedings of the IEEE 31st International Conference on Tools with Artificial Intelligence, Portland, OR, USA, 4–9 November 2019; pp. 1118–1123.
40. Miki, S.; Yamamoto, D.; Ebara, H. Applying deep learning and reinforcement learning to traveling salesman problem. In Proceedings of the International Conference on Computing, Electronics & Communications Engineering (iCCECE 2018), Southend, UK, 16–17 August 2018; pp. 65–70.
41. Bentley, J.J. Fast algorithms for geometric traveling salesman problems. *Orsa J. Comput.* **1992**, *4*, 387–411. [CrossRef]
42. Wang, S.; Rao, W.; Hong, Y. A distance matrix based algorithm for solving the traveling salesman problem. *Oper. Res.* **2018**, *20*, 1–38. [CrossRef]

43. Jackovich, P.; Cox, B.; Hill, R.R. Comparing greedy constructive heuristic subtour elimination methods for the traveling salesman problem. *J. Def. Anal. Logist.* **2020**, *4*, 167–182. [CrossRef]
44. Chuang, C.C.; Su, S.F.; Hsiao, C.C. The annealing robust backpropagation (ARBP) learning algorithm. *IEEE Trans. Neural Netw.* **2000**, *11*, 1067–1077. [CrossRef]
45. Miller, J.N. Tutorial review—Outliers in experimental data and their treatment. *Analyst* **1993**, *118*, 455–461. [CrossRef]
46. Bardach, E. The extrapolation problem: How can we learn from the experience of others? *J. Policy Anal. Manag.* **2004**, *23*, 205–220. [CrossRef]
47. Hougardy, S.; Schroeder, R.T. Edge elimination in TSP instances. In *International Workshop on Graph-Theoretic Concepts in Computer Science*; Springer: Cham, Switzerland, 2014; pp. 275–286.
48. Lemaître, G.; Nogueira, F.; Aridas, C.K. Imbalanced-learn: A python toolbox to tackle the curse of imbalanced datasets in machine learning. *J. Mach. Learn. Res.* **2017**, *18*, 559–563.
49. Applegate, D. Concorde—A Code for Solving Traveling Salesman Problems. Available online: http://www.math.princeton.edu/tsp/concorde.html (accessed on 8 September 2021).
50. Colquhoun, D. The reproducibility of research and the misinterpretation of p-values. *R. Soc. Open Sci.* **2017**, *4*, 171085. [CrossRef] [PubMed]
51. Vitali, T.; Mele, U.J.; Gambardella, L.M.; Montemanni, R. Machine Learning Constructives and Local Searches for the Travelling Salesman Problem. *arXiv* **2021**, arXiv:2108.00938.
52. De Boer, P.T.; Kroese, D.P.; Mannor, S.; Rubinstein, R.Y. A tutorial on the cross-entropy method. *Ann. Oper. Res.* **2005**, *134*, 19–67. [CrossRef]
53. Williams, R.J. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Mach. Learn.* **1992**, *8*, 229–256. [CrossRef]
54. Van Rossum, G.; Drake, F.L., Jr. *Python Tutorial*; Centrum voor Wiskunde en Informatica Amsterdam: Amsterdam, The Netherlands, 1995; Volume 620.
55. Paszke, A.; Gross, S.; Chintala, S.; Chanan, G.; Yang, E.; DeVito, Z.; Lin, Z.; Desmaison, A.; Antiga, L.; Lerer, A. Automatic differentiation in pytorch. In Proceedings of the 31st Conference on Neural Information Processing System, Long Beach, CA, USA, 8 December 2017.