Reinforcement Learning and Additional Rewardsfor the Traveling Salesman Problem

Umberto Junior Mele, Xiaochen Chou, Luca Maria Gambardella
Dalle Molle Institute for Artificial Intelligence
IDSIA-USI/SUPSI
Manno, Switzerland
e-mail: {umbertojunior.mele, xiaochen, luca}@idsia.ch

Roberto Montemanni
Department of Sciences and Methods for Engineering
University of Modena and Reggio Emilia
Reggio Emilia, Italy
e-mail: roberto.montemanni@unimore.it

***Abstract*—A comprehensive literature on the Traveling Salesman Problem (TSP) is available, and this problem has become a valuable benchmark to test new heuristic methods for general Combinatorial Optimisation problems. For this reason, recently developed Deep Learning-driven heuristics have been tried on the TSP. These Deep Learning frameworks use the city coordinates as inputs, and are trained using reinforcement learning to predict a distribution over the TSP feasible solutions. The aim of the present work is to show how easy-to-calculate Combinatorial Optimization concepts can improve the performances of such systems. In particular, we show how passing Minimum Spanning Tree information during training can lead to significant improvements to the quality of TSP solutions.**

As a side result, we also propose a Deep Learning architecture able to predict in real time the optimal length of a TSP instance.

The proposed architectures have been tested on random 2D Euclidean graphs with 50 and 100 nodes, showing significant results.

***Keywords*-Deep Learning, Reinforcement Learning, Traveling Salesman Problem, Minimum Spanning Tree Problem, Combinatorial Optimization.**

I. Introduction

In recent years, a growing interest in Machine Learning (ML) and Deep Learning (DL) has been experienced, motivated by the successful results achieved in several fields [1], [2], [3]. A natural question is how ML techniques can be used to successfully tackle NP-hard Combinatorial Optimization (CO) problems. Often the state-of-the-art solving methodologies for such problems are characterized by long computation times, and in some other cases the objective functions can not be well defined mathematically, making formal theoretical methods difficult to apply. Machine Learning approaches seem to be a good choice to deal with these cases effectively [4], thanks to the fast solution times (after training) characterizing them and their data-driven nature.

In this work we start from one of the most promising Machine Learning approaches for the TSP, presented in [5] by Deudon et al. and based on Geometric Deep Learning (GDL) in a Reinforcement Learning (RL) framework [6], [7]. We show how some Combinatorial Optimization concepts, such as the polynomially-computable Minimum Spanning Trees in our case, can be used to improve the effectiveness of the method, leading to better results. The final aim of this research is to understand how Machine Learning methods and Combinatorial Optimization concepts can be hybridized in an enriched framework able to tackle other Combinatorial Optimization problems efficiently.

Umberto Junior Mele and Xiaochen Chou were supported by the Swiss National Science Foundation through grants 200020-182360: "Machine Learning for Combinatorial Optimization".

Any CO problem can be stated as the task of finding the optimal object from a finite set of objects. In the TSP the objective is to find a tour of minimum cost among all feasible tours. It is clear that searching through all possible tours is not viable from a computational point of view. Therefore, smarter approaches are needed. Heuristic algorithms such as the one described in [8] have been proposed to solve the TSP. Generally, these methods are very efficient and effective. Note that our goal here is not to compete against them, but to propose alternative methods that in perspective can provide high-quality approximate solutions for the TSP in pseudo-linear time, and eventually be generalized to other Combinatorial Optimization problems as well. It is worth noting that there are also exact methods available for the TSP, such as Concorde [9], that have been proved being able to find the optimal solution for a variety of TSP instances, although the computation time can be prohibitive for larger instances.

The paper is organized as follows. In Section I-A we briefly discuss the literature related to this work. In Section II we describe the TSP; in Section III the RL framework for this problem is stated; Section IV is devoted to the method description and the insights behind it. Finally, in Section V we show and discuss our experimental results for the solver and the optimal length prediction, providing a comparisons with other RL solvers. Conclusions are finally drawn in Section VI.

A. Related work

Vinyals et al. [10] introduced with Pointer Networks the first Deep Learning architecture oriented to Combinatorial Optimization problems. Their initial model was able to output a distribution on the permutation of the input, and it was trained offline to solve the Euclidean TSP, supervised by example solutions. To train this model without supervised solutions, Bello et al. [11] proposed an *Actor-Critic* algorithm considering each instance as a training sample and using the cost (tour length) of a sampled solution to estimate the policy gradients employing Monte-Carlo (Section IV-C). Vinyals et al. [12] improved their previous model using extra depth for the decoder through a function called the *glimpse* (Section IV-B), which creates a representation for the whole input instance, encoding even the instance topology. Starting from the same work, Nazari et al. [13] provided some technical contribution by replacing the previous decoder with an element-wise projections (*Pointing mechanism*), in order to ease the update of the nodes embedding (Section IV-B). Nowak et al. [14] used *Graph Neural Networks* in a supervised manner, while Kaempfer & Wolf [15] were the first to use the famous architecture called *Transformer* to output solutions for to the multiple TSP. Deudon et al. [5] and Kool et al. [16] presented models for Euclidean TSP that use *Transformer* and *Actor-Critic* together, with the latter one achieving state-of-the-art results for Deep Learning-based heuristics for the TSP. The networks are however very similar. An interesting direction to extend the combinatorial generalization is given by Battaglia et al. [17], where they suggest a framework that could be useful to overpass scalability problems. An example of the use of Machine Learning techniques within Combinatorial Optimization methods can be finally found in Montemanni et al. [18].

II. Problem Definition

Given a set of cities and distance between every pair of cities, the objective of the Travelling Salesman Problem (TSP) is to find the shortest possible route that visits each city exactly once and returns to the starting point.

We denote with $V = \{1, \ldots, n\}$ a set of n cites or nodes. Let d_{ij} be the distance from city i to city j, and $x_{ij} = \{0, 1\}$ a variable, which takes value 1 if the path goes from city i to city j, otherwise 0. Each city is arrived at from exactly one other city, and is a departure to exactly one other city. A feasible solution X is a matrix $n \times n$ of x_{ij}, and the set of feasible solutions is denoted as $\mathcal{X}$.

The objective function is to minimize the tour length as described in Eq. 1.

$$\min \quad \sum_{i=1}^{n} \sum_{j>i}^{n} \quad d_{ij}\, x_{ij} \quad , x_{ij} \in \{0,1\}, X \in \mathcal{X} \tag{1}$$

As we said in the introduction, we employ also the Minimum Spanning Tree (MST), which is a tree that spans all the cities using just the shortest possible distances. It is well known that for each Euclidean TSP instance there exist at least one optimal MST, and it can be found in polynomial time [19]. Note that computing an optimal TSP is instead NP-hard. The fact that somehow the MST is a good approximation of the TSP solution (Fig. 1) is exploited during the RL framework, as shown in Sections III and IV-C.

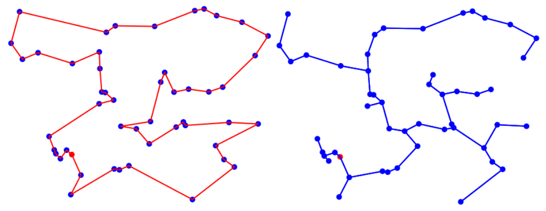
Fig. 1: Comparison of the Traveling Salesman Problem solution and the Minimum Spanning Tree for a same instance.

III. A Markov Decision Process for the TSP

From the RL perspective, the TSP problem can be stated according to the Markov Decision Process (MDP) theory [7], [20]. It describes the behaviour of an agent that makes actions/movements in a certain environment in order to maximize its rewards and minimize its sanctions during time. Such process of actions is supposed to be Markovian [21], meaning that the agent does not need any additional information coming from its future actions to make the next movement, since all the useful information is in the actual state which is the result of the past moves. With this in mind, we describe the TSP using the MDP notation:

- t is an index that refers to the "*time*" of the process, for the TSP $t \in \{0, 1, ..., n-2\}$ since the agent has an initial moment 0 where it needs to choose a starting city, then it proceeds choosing cities until time $n-2$ where it is constrained to select last left city and return to the starting city.
- the "*state vector*" $\mathbf{s_t} \in \mathbf{S}$ is used by the agent to describe the environment at time t. Thus, it has to take into account all the useful information needed to solve the task, such as the topology of the TSP problem that is currently solving, the cities visited, those to visit and the current one. $\mathbf{S}$ is the set of all feasible states.
- m_t refers to the "*movement*" made by the agent at time t; in our case it is stated by a single integer that points to the selected city to visit and it is forbidden to visit the same city twice, then the constraint $m_t \in \mathbf{M_t}$ is needed, where $\mathbf{M_t}$ is the set of available movements at time t. Note, given that the movements $m_t = j$ and $m_{t-1} = i$ have occurred, then we can say that $x_{ij} = 1$ is in the solution.
- "*rewards*" and "*sanctions*" values are provided to the agent to let it evaluate positively or negatively his movements. Thus given the complete route traversed by the agent, the adjacency matrix X that describe this route is computed, then reward and sanction functions are measured. The sanction was given by the route length $\mathcal{S}_{\mathrm{L}}$ found by the agent:

$$\mathcal{S}_{\mathrm{L}}(X) = \sum_{i} \sum_{j>i} d_{ij}\, x_{ij} \tag{2}$$

A first contribution of the present work is using an acknowledgement given by the MST information $\mathcal{R}_{\mathrm{MST}}$

as additional reward value to the agent:

$$\mathcal{R}_{\text{MST}}(X) = \frac{\sum_{\{i,j\}\in\mathcal{MST}} x_{ij}}{n}$$

where $\mathcal{MST}$ is the set of edges of an optimal MST solution.
The rationale is to compute the number of edges in common between the current TSP solution and the MST (this quantity is divided by n to normalize).
When the $\mathcal{R}_{\text{MST}}$ contributions is considered, the total reward to be maximized is given as $\mathcal{R}_{\text{tot}} = \mathcal{R}_{\text{MST}} - \mathcal{S}_{\text{L}}$, since the agent objective is to minimize the sanction while maximizing the MST reward.

- the "*stochastic policy*" function $\pi_\theta(\cdot\,|\,\mathbf{s_t})$ states precisely the rules that determine the agent behaviour. For the TSP, it defines a probability distribution on the feasible movements given the state vector at time t. Such a function is approximated by an artificial neural network. Once the network's structure is defined, the set of parameters $\boldsymbol{\theta}$ – which collects all the weights for each artificial neuron in the network – identifies a unique policy function, where $\pi_\theta(m_t = i|\,\mathbf{s_t}):\ \mathbf{M_t}\times\mathbf{S}\rightarrow[0,1]$.
- the "*transition state rules*" are the ones that define the environment. For the TSP these rules are deterministic, since a new state $\mathbf{s_{t+1}}$ is completely defined after the previous state movement m_t has been made.

Accordingly to this notation, and given that we input our model as a $n\times 2$ matrix $\mathbf{P}$ containing the coordinates of the 2D positions for the n cities in the problem, the goal is to learn by means of RL the parameters θ defining the probability function $f_\theta(\cdot|\,\mathbf{P})$ on the feasible solution space $\mathcal{X}$. With the probability function it is possible to quickly sample solutions, and in such a way that $X\in\mathcal{X}$. Such probability function is defined by means of the agent policy function:

$$f_\theta(X|\,\mathbf{P}) = \prod_{t=1}^{n}\pi_\theta(m_t|\,\mathbf{s_t}) \qquad \forall m_t\in\{1,...,n\} \tag{3}$$

The desired policy is the one minimizing the expected length of the sampled tours for a given TSP instance:

$$\text{argmin}_\theta\ \mathbb{E}_{f_\theta}\Big[\mathcal{R}_{\text{tot}}\Big] \tag{4}$$

To achieve this task, we employ the REINFORCE trick [22], which will be described later in Section IV.C.

IV. Methods

To increase the network's ability in creating internal representations useful to solve the task, we exploit, as previous works [5], [11], [16], the Encoder-Decoder Deep Learning scheme.

The "*Encoder*" duty is to map the input cities positions $\mathbf{P} = \{\mathbf{p_1},\mathbf{p_2},...,\mathbf{p_n}\}$ into a feature space used to represent the knowledge through feature representations of each city $\mathbf{E} = \{\mathbf{e_1},\mathbf{e_2},...,\mathbf{e_n}\}$. Such matrix $\mathbf{E}$ has dimension $n\times d_e$, and in the implementation has $d_e = 128$, similarly to Deudon et al. [5]. Then the "*Decoder*" is in charge of processing these representations and compute the desired solution $\mathbf{X}$, through the policy function as in Eq. (3).

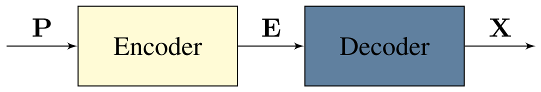

Fig. 2: the Encoder-Decoder pipeline

Both Encoder and Decoder parameters are learned by means of the RL training algorithm called "*Policy Gradient*" [23] (Section IV-C).

A. The Encoder

A relevant aspect for an encoder that needs to deal with CO problems and in particular with the TSP, is that it needs to handle data represented troughs graphs. Hence, an obvious choice from the ML point of view, is to address the problem using Graph Neural Networks (GNN) [6]. These techniques are quite recent and appear to have fascinating properties such as "*nodes order invariance*" and "*local affinity representation*", and here they are used to ease the artificial neural network in creating meaningful representations for each node in the instance.

Nodes order invariance is detailed by the learner's ability to output the same solutions even if the inputs are modified with any permutation on the nodes order. Such property is significant for graphs since there is no order between the vertices of a graph, and therefore there will be an unwanted behaviour if our Encoder behaves differently for different permutations.

Local affinity for graphs is the perspective that connected nodes in a graph should be similar or close in some sense, meaning that they have some feature in common. For this reason GNNs are creating features representations that are similar or close in the space for connected nodes.

Taking into account these properties, we briefly show how such Encoder network is composed. After an initial linear projection of the position matrix $\mathbf{P}$ in the higher dimensional space $\mathbf{E}$, the Encoder is strongly based on an adaptation of the "*Transformer*" stack [24] suited for graphs. A stack in the DL terminology is a set of different layers, and here the adaptation is created by exploiting the "*Graph Attention layer*" (GAT) [25] instead of the usual "*Attention layer*" [26], since these mechanism are very similar, but the first one is specialized for graphs.

The insight beside the "*attention*" mechanism, is that given a certain context, not all the entities in such a context are important. To focus on just the dominant ones, some weights are then given to such entities. In our case, the aim is to create a context representations for each node i. These representations are created taking into consideration information arising from the other nodes j in the neighborhood of each node i. Practically, the context vector is just a simple weighted average of the features in the context, where the average's coefficients are given by a function of the entities in the context. For more details, we address the interested reader to Veličković et al. [25].

Hence, in order to replicate all the transformation in the representation space to reach the final matrix $\mathbf{E}$ and to replicate totally such adapted "Transformer" stack are needed the following layers:

- *Multi-Head Graph Attention* (MH-GAT) is the layer in the pipeline in charge of processing the nodes internal representations in order to enclose the context knowledge. It is composed by several GAT layers employed in parallel and finally averaged, such that it stabilize the learning process and it gives a higher degree of freedom to learn useful representations. As Fig. 3 shows, this layer takes as input all the context of the node (in this case node 1 and its neighbors in green), and returns a representation for such node (node 1 in blue). To have a deeper insight of the mathematical details of such layer we suggest to read Vaswani et al. [24] and Veličković et al. [25].

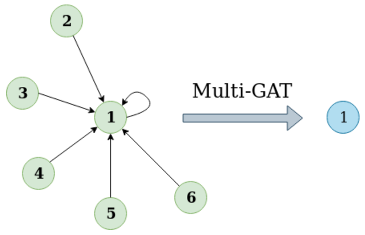

Fig. 3: Multi-Head Attention for node 1

- *Add & Norm* is the connection layer, its functionality is to ease the gradient back-propagation to reach better the lowest layers in the model. The term *Add* is referred to the residual connection [27], [28], and is just the addition of the inputs of a given transformation with its output. Then, the batch *Normalization* operation is finally applied in pipeline to the output of the residual operation to avoid the internal covariate shift problem [29].

- *Vertex-Wise Feed Forward Network* (VW-FFN) consists of two linear transformations detached by an non-linear activation function called ReLU[1]. The VW-FFN is applied to each node representation, threfore Vertex-Wise. It creates a representation for each through a selection of the features that are useful to solve the task.

So, given these layers, we can summarize the "*Transformer*" stack with the pipeline scheme of Fig. 4. The stack described is repeated three times, and the final output of the Encoder is the matrix $\mathbf{E}$.

[1] $ReLU(x) = max(0, x)$

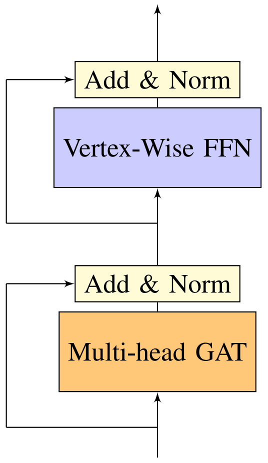

Fig. 4: The Transformer pipeline

B. The Decoder

As briefly discussed in Section III, the decoder block needs to provide a policy function $\pi_\theta(\cdot|\,\mathbf{s_t})$, which is used to create solutions $\mathbf{X}$ as shown in Eq. (3). First, we needed to define the state vector $\mathbf{s_t}$ that contains all the information useful to select the next action m_t. We can summarize them as follows:

- the graph information vector s_{graph} is fixed with respect to time t, but changes between different TSP instances. It is defined as a weighted average where the weights are learned through a function similarly to "*attention*", which is called "*glimpse*" (for more details we refer the interested reader to Vinyals et al [12]).
- an encoding of the last three movements s_{past} that helps the agent to keep in memory the direction taken. Such encoding is simply a linear projection, in pipeline with the ReLU activation, of the last tree visited cities representations $[\mathbf{e_{m_t}}, \mathbf{e_{m_{t-1}}}, \mathbf{e_{m_{t-2}}}]$ (details can be found in [5])
- a *mask* on the visited cities to prevent the agent to carry out infeasible moves.

Finally, this information is concatenated in the vector $\mathbf{s_t}$

$$\mathbf{s_t} = [s_{\text{graph}},\ s_{\text{ past}}]$$

and given as input to the "Pointing Mechanism", which is in charge of selecting the next city. This is achieved by sampling from the output distribution of the Pointing Mechanism itself, as shown in Fig. 5. The mechasism takes in input the state $\mathbf{s_t}$ and the unvisited cities representations $\mathbf{e_i}$, then provides as output a distribution on these cities (details in Nazari et al. [13]).

C. Training phase

The DL architecture previously described is trained via the optimizer ADAM [30], and to use such algorithm, it is necessary to compute the gradients ∇_θ of the reward function defined in Eq. 4, with respect to all the parameters in the set θ:

$$\nabla_\theta \mathbb{E}_{f_\theta}\left[\mathcal{R}_{tot}(\mathbf{X})\right] \tag{5}$$

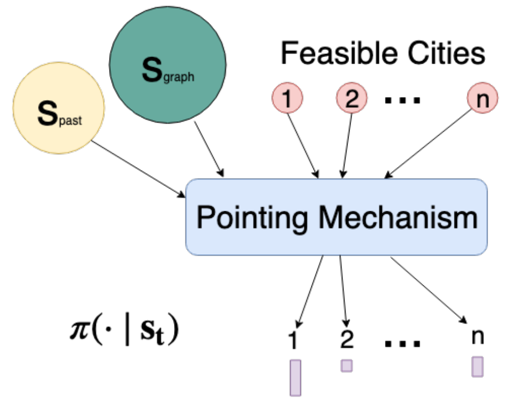

Fig. 5: The Pointing mechanism

Then, to avoid the derivation of the expectation in Eq 5, the REINFORCE approximation [22] is employed. It is a derivation trick that allow to rewrite the original equation in such a way that the gradients are now computed with respect to the function f_θ, and therefore with respect to the artificial neural network approximation of the policy function π_θ (recall Eq. 3):

$$\mathbb{E}_{f_\theta}\left[\mathcal{R}_{tot}(\mathbf{X})\cdot\nabla_\theta \log\left(f_\theta(\mathbf{X}|\mathbf{P})\right)\right] \tag{6}$$

Then, as introduced in Bello et. al [11], another level of approximation is employed, starting from Eq. (6). It uses the Actor-Critic framework [31], that is a type of REINFORCE *with baseline* [32], and the aim is to reduce the gradient variance and therefore increase the speed of learning. We extend their idea [11], using the sanction $\mathcal{S}_{\text{L}}$ coming from the length, but joining it with the additional reward $\mathcal{R}_{\text{MST}}$ given by the MST supervision. These functions are subtracted from the baselines, in such a way to work with the effective value for the reward and the sanction:

$$\mathbb{E}_{f_\theta}\left[\left(\boldsymbol{\Delta}\mathcal{R}_{\text{MST}}-\Delta\mathcal{S}_{\text{L}}\right)\cdot\nabla_\theta \log\left(f_\theta(\mathbf{X}|\mathbf{P})\right)\right] \tag{7}$$

where the delta functions are defined as follows:

$$\boldsymbol{\Delta}\mathcal{R}_{\text{MST}}=\mathcal{R}_{\text{MST}}(\mathbf{X})-b_\delta(\mathbf{P}) \tag{8}$$

$$\Delta\mathcal{S}_{\text{L}}=\mathcal{S}_{\text{L}}(\mathbf{X})-b_\gamma(\mathbf{P}) \tag{9}$$

with $b_\delta(\mathbf{P})$ and $b_\gamma(\mathbf{P})$ being the *baseline* functions (also known as critic networks), and $\Delta\mathcal{R}_{\text{MST}}$ with $\Delta\mathcal{S}_{\text{L}}$ represent the effective reward with the effective sanction. The insight is to reduce the variance of $\mathcal{R}_{\text{MST}}$ and $\mathcal{S}_{\text{L}}$ by subtracting a baseline value from them. Such a "*baseline*" represents the normal reward and sanction for the current agent (or Actor) behavior. Therefore the environment is recompensing actions corresponding to something new.

Both critics are defined as Feed-Foward Networks (FFNs) [33] with input the s_{graph} vector; such networks are similar to the one shown in Section IV-A, with the main difference that they aim to predict the values $\mathcal{R}_{\text{MST}}$ and $\mathcal{S}_{\text{L}}$ instead of creating representations. They are trained simultaneously to the Actor, with the goal of minimizing their respectively delta functions (Eqs. 8 and 9).

Finally, the expectation in Eq. (7) is estimated using Monte Carlo sampling, with samples of solutions $\mathbf{X}$ from the Actor policy; and the gradient is back-propagated allowing the parameters to be trained on millions of TSP instances with the only supervision of the MST.

V. Experiments and Results

To evaluate and train the framework of Deudon et al. [5] and our extensions, we used random Euclidean TSP instances. The nodes $\mathbf{p_i}$ of these TSP instance have been drawn uniformly at random from a 2 dimensional unit square. All the experiments were handled using Tensorflow 1.13.1. [34] on the following hardware:

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

All the hyper-parameters for Encoder and Decoder are the same as in [5], and we use their code "*Encode-Attend-Navigate*" (EAN) available online[2] as a baseline for our extensions.

Results are compared in terms of optimality gap with respect to the optimal length:

$$Gap=\frac{|\mathcal{S}_{\text{L}}(\mathbf{X})-\mathcal{S}_{\text{L}}(\dot{\mathbf{X}})|}{\mathcal{S}_{\text{L}}(\dot{\mathbf{X}})} \tag{10}$$

where $\mathcal{S}_{\text{L}}(\cdot)$ is the tour length function defined in Eq. (2), $\mathbf{X}$ is the tour under evaluation, and $\dot{\mathbf{X}}$ is the optimal tour retrieved by the exact TSP solver Concorde [35].
To check the statistical significance of our extension – that will be referred to as MST in the reminder of the paper – with respect to the previous network EAN, we have evaluated the p-value using the Z-test [36]; whit null hypothesis:

$\mathbf{H_0}$: *"on average, the the optimality gap Gap_{MST} of model MST is not lower than Gap_{EAN}, the optimality gap of EAN"*. Noting that our method reduce significantly the uncertainty with respect to the other predictors, our proposal for future work is to know if such estimator could be extended to TSPs with different dimension, and we are even curious to know if the approach could be used jointly with a reconstruction algorithms (i.e. the $A^\star$ search [37]) to provide solutions.

$$\mathbb{E}\left[Gap_{MST}-Gap_{EAN}\right]\geq 0$$

where it is assumed (worst case) that the distribution of this difference is normal with 0 average, and that the standard deviation is equal to the empirical one.

[2]https://github.com/MichelDeudon/encode-attend-navigate

TABLE I: Results for the prediction of the TSP optimal tour.

Models	n = 50		n = 100	
	time	gap	time	gap
EAN	0.80 s	1.239 % ± 1.181 %	1.57 s	11.238 % ± 1.734 %
MST	**0.80 s**	**1.152 % ± 1.128 %**	**1.57 s**	**10.954 % ± 1.628%**

TABLE II: Tour Prediction Evaluation.

Models	n = 50	
	time	gap
Critic	**0.004 s**	**1.06 % ± 0.80 %**
Average	–	3.48 % ± 2.61 %
Bootstrap	–	4.77 % ± 3.63 %

A. *Prediction of the TSP optimal tour*

Similarly to Deudon et al. [5], we have trained our architecture on $5.12 \cdot 10^6$ TSP instances with 50 cities for two epochs, with a total training time of 4 hours for each method.

Then we tested the models trained on instances with 50 nodes on cases with 50 and 100 nodes (to see if the networks are able to generalize). For each case, the same benchmark dataset of 1000 random Euclidean TSP has been used to compare the models. For each instance and each method, the best solution over 5000 sampled ones, obtained using the output distribution of the networks and Eq. 3, is considered.

The results are presented in Table I, where our extension MST is compared to EAN network (no 2opt) in terms of average time to solve one instance and average gap (Eq. 10) plus standard deviation gap. Note that the computation time is very small due to parallelization, and it is clear that our extension achieve an interesting improvement in term of gap for both dimension, keeping unchanged the time consumption, since the network structure is unchanged. The result clearly indicates that using easy-to-compute CO concepts can improve the performance of DL-driven heuristics.

With the values shown in Table I, it is achieved a p-value equal to $6.4 \cdot 10^{-5}$ for the $n = 50$ case, whereas a value of $3 \cdot 10^{-15}$ for the $n = 100$ one. Therefore the null hypothesis is rejected for both cases, and our improvements are significant.

B. *Prediction of the TSP optimal tour length*

A second contribution that we introduce, is the idea of using a critic architecture $b_\eta(\mathbf{P})$ to predict the optimal length $\mathcal{S}_{\mathrm{L}}(\dot{\mathbf{X}})$ for a given instance defined by the cities position matrix $\mathbf{P}$. The problem is known to be NP-hard, since it requires usually to solve the TSP, and the interest here is to provide quick and good approximation to such a value.

The network is trained on $5.12 \cdot 10^6$ TSP 50 instances for five epochs, and the resulting model is tested on 1000 instances with 50 cities, as reported in Table II. In the table the average gap (Eq. 10) is shown together with standard deviation, and the average time required by the method we propose, *Critic*, to predict the value for one instance. Due to the lack or results in the literature, the method is compared with the empirical distribution of the optimal lengths for the Euclidean TSP 50 instances, trough the *Average* value predictor, which is used to predict all the lengths in the test set, and the *Bootstrap* [38], that samples, each time, from the empirical distribution a length and it uses like prediction for the instance.

Noting that our method reduce significantly the uncertainty with respect to the other predictors, our proposal for future work is to know if such estimator could be extended to TSPs with different dimension, and we are even curious to know if the approach could be used jointly with reconstruction algorithms (i.e. the $A^\star$ search [37]) to provide solutions.

VI. Conclusions and future work

The aim of the present work was to show how Combinatorial Optimization concepts can be used to improve the accuracy of Machine Learning techniques when the latter are used to learn heuristic methods for optimization tasks. In particular, we have considered the NP-hard Traveling Salesman Problem, and we have used information extracted from the polynomially-computable Minimum Spanning Tree problem to improve the learning accuracy of a Machine Learning algorithm recently appeared in the literature.

A side-contribution has been also to show that Machine Learning techniques can be devised to estimate with high precision, and in negligible time, the cost of an optimal Traveling Salesman tour.

The current solver presents however some limitations. It works only with (small) fixed-size instances where distances are strictly Euclidean. Our future work will address these limitations.

References

[1] Y. Wu, M. Schuster, Z. Chen, Q. V. Le, M. Norouzi, W. Macherey, M. Krikun, Y. Cao, Q. Gao, K. Macherey, J. Klingner, A. Shah, M. Johnson, X. Liu, L. Kaiser, S. Gouws, Y. Kato, T. Kudo, H. Kazawa, K. Stevens, G. Kurian, N. Patil, W. Wang, C. Young, J. Smith, J. Riesa, A. Rudnick, O. Vinyals, G. Corrado, M. Hughes, and J. Dean, "Google's neural machine translation system: Bridging the gap between human and machine translation," *CoRR*, vol. abs/1609.08144, 2016. [Online]. Available: http://arxiv.org/abs/1609.08144

[2] A. van den Oord, S. Dieleman, H. Zen, K. Simonyan, O. Vinyals, A. Graves, N. Kalchbrenner, A. Senior, and K. Kavukcuoglu, "Wavenet: A generative model for raw audio," in *Arxiv*, 2016. [Online]. Available: https://arxiv.org/abs/1609.03499

[3] J. S. Chung, A. W. Senior, O. Vinyals, and A. Zisserman, "Lip reading sentences in the wild," *CoRR*, vol. abs/1611.05358, 2016. [Online]. Available: http://arxiv.org/abs/1611.05358

[4] Y. Bengio, A. Lodi, and A. Prouvost, "Machine learning for combinatorial optimization: a methodological tour d'horizon," *CoRR*, vol. abs/1811.06128, 2018. [Online]. Available: http://arxiv.org/abs/1811.06128

[5] M. Deudon, P. Cournut, A. Lacoste, Y. Adulyasak, and L.-M. Rousseau, “Learning heuristics for the tsp by policy gradient,” in *Integration of Constraint Programming, Artificial Intelligence, and Operations Research*. Cham: Springer International Publishing, 2018.

[6] M. M. Bronstein, J. Bruna, Y. LeCun, A. Szlam, and P. Vandergheynst, “Geometric deep learning: Going beyond euclidean data,” *IEEE Signal Processing Magazine*, vol. 34, no. 4, pp. 18–42, July 2017.

[7] C. Szepesvari, *Algorithms for Reinforcement Learning*. Morgan and Claypool Publishers, 2010.

[8] K. Helsgaun, “An effective implementation of the lin-kernighan traveling salesman heuristic,” *European Journal of Operational Research*, vol. 126, pp. 106–130, 2000.

[9] D. Applegate, W. J. Cook, S. Dash, and A. Rohe, “Solution of a min-max vehicle routing problem,” *INFORMS Journal on Computing*, vol. 14, pp. 132–143, 2002.

[10] O. Vinyals, M. Fortunato, and N. Jaitly, “Pointer networks,” in *Advances in Neural Information Processing Systems 28*. Curran Associates, Inc., 2015, pp. 2692–2700. [Online]. Available: http://papers.nips.cc/paper/5866-pointer-networks.pdf

[11] I. Bello, H. Pham, Q. V. Le, M. Norouzi, and S. Bengio, “Neural combinatorial optimization with reinforcement learning,” *CoRR*, vol. abs/1611.09940, 2016. [Online]. Available: http://arxiv.org/abs/1611.09940

[12] O. Vinyals, S. Bengio, and M. Kudlur, “Order matters: Sequence to sequence for sets,” in *International Conference on Learning Representations*, 2016. [Online]. Available: http://arxiv.org/abs/1511.06391

[13] M. Nazari, A. Oroojlooy, L. Snyder, and M. Takac, “Reinforcement learning for solving the vehicle routing problem,” in *Advances in Neural Information Processing Systems 31*. Curran Associates, Inc., 2018, pp. 9839–9849. [Online]. Available: http://papers.nips.cc/paper/8190-reinforcement-learning-for-solving-the-vehicle-routing-problem.pdf

[14] A. Nowak, S. Villar, A. S. Bandeira, and J. Bruna, “Revised note on learning quadratic assignment with graph neural networks,” in *2018 IEEE Data Science Workshop (DSW)*, June 2018, pp. 1–5.

[15] Y. Kaempfer and L. Wolf, “Learning the multiple traveling salesmen problem with permutation invariant pooling networks,” *CoRR*, vol. abs/1803.09621, 2018.

[16] W. Kool, H. van Hoof, and M. Welling, “Attention, learn to solve routing problems!” in *International Conference on Learning Representations*, 2019. [Online]. Available: https://openreview.net/forum?id=ByxBFsRqYm

[17] P. Battaglia, J. B. C. Hamrick, V. Bapst, A. Sanchez, V. Zambaldi, M. Malinowski, A. Tacchetti, D. Raposo, A. Santoro, R. Faulkner, C. Gulcehre, F. Song, A. Ballard, J. Gilmer, G. E. Dahl, A. Vaswani, K. Allen, C. Nash, V. J. Langston, C. Dyer, N. Heess, D. Wierstra, P. Kohli, M. Botvinick, O. Vinyals, Y. Li, and R. Pascanu, “Relational inductive biases, deep learning, and graph networks,” *CoRR*, vol. abs/1806.01261, 2018. [Online]. Available: http://arxiv.org/abs/1806.01261

[18] R. Montemanni, F. D’Ignazio, X. Chou, and L. M. Gambardella, “Machine learning and monte carlo sampling for the probabilistic orienteering problem,” in *2018 Joint 10th International Conference on Soft Computing and Intelligent Systems and 19th International Symposium on Advanced Intelligent Systems*. IEEE, 2018, pp. 14–18.

[19] M. L. Fredman and R. E. Tarjan, “Fibonacci heaps and their uses in improved network optimization algorithms,” *J. ACM*, vol. 34, no. 3, pp. 596–615, Jul. 1987. [Online]. Available: http://doi.acm.org/10.1145/28869.28874

[20] A. N. Burnetas and M. N. Katehakis, “Optimal adaptive policies for markov decision processes,” *Math. Oper. Res.*, vol. 22, no. 1, pp. 222–255, Feb. 1997. [Online]. Available: http://dx.doi.org/10.1287/moor.22.1.222

[21] A. Markov, *Theory of Algorithms*, ser. TT 60-51085. Academy of Sciences of the USSR, 1954. [Online]. Available: https://books.google.ch/books?id=mKm1swEACAAJ

[22] R. J. Williams, “Simple statistical gradient-following algorithms for connectionist reinforcement learning,” *Machine Learning*, vol. 8, no. 3, pp. 229–256, May 1992. [Online]. Available: https://doi.org/10.1007/BF00992696

[23] R. S. Sutton, D. McAllester, S. Singh, and Y. Mansour, “Policy gradient methods for reinforcement learning with function approximation,” in *Proceedings of the 12th International Conference on Neural Information Processing Systems*, ser. NIPS’99. Cambridge, MA, USA: MIT Press, 1999, pp. 1057–1063. [Online]. Available: http://dl.acm.org/citation.cfm?id=3009657.3009806

[24] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. u. Kaiser, and I. Polosukhin, “Attention is all you need,” in *Advances in Neural Information Processing Systems 30*. Curran Associates, Inc., 2017, pp. 5998–6008. [Online]. Available: http://papers.nips.cc/paper/7181-attention-is-all-you-need.pdf

[25] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Liò, and Y. Bengio, “Graph Attention Networks,” *International Conference on Learning Representations*, 2018. [Online]. Available: https://openreview.net/forum?id=rJXMpikCZ

[26] D. Bahdanau, K. Cho, and Y. Bengio, “Neural machine translation by jointly learning to align and translate,” *CoRR*, vol. abs/1409.0473, 2015.

[27] K. He, X. Zhang, S. Ren, and J. Sun, “Deep residual learning for image recognition,” *CoRR*, vol. abs/1512.03385, 2015. [Online]. Available: http://arxiv.org/abs/1512.03385

[28] R. K. Srivastava, K. Greff, and J. Schmidhuber, “Highway networks,” *CoRR*, vol. abs/1505.00387, 2015.

[29] S. Ioffe and C. Szegedy, “Batch normalization: Accelerating deep network training by reducing internal covariate shift,” *CoRR*, vol. abs/1502.03167, 2015. [Online]. Available: http://arxiv.org/abs/1502.03167

[30] D. P. Kingma and J. Ba, “Adam: A method for stochastic optimization,” 2014, cite arxiv:1412.6980Comment: Published as a conference paper at the 3rd International Conference for Learning Representations, San Diego, 2015. [Online]. Available: http://arxiv.org/abs/1412.6980

[31] V. Konda, “Actor-critic algorithms,” Ph.D. dissertation, Massachusetts Institute of Technology, Cambridge, MA, USA, 2002, aAI0804543.

[32] T. Zhao, H. Hachiya, G. Niu, and M. Sugiyama, “Analysis and improvement of policy gradient estimation,” in *Advances in Neural Information Processing Systems 24*, J. Shawe-Taylor, R. S. Zemel, P. L. Bartlett, F. Pereira, and K. Q. Weinberger, Eds. Curran Associates, Inc., 2011, pp. 262–270. [Online]. Available: http://papers.nips.cc/paper/4264-analysis-and-improvement-of-policy-gradient-estimation.pdf

[33] J. Schmidhuber, “Deep learning in neural networks: An overview,” *CoRR*, vol. abs/1404.7828, 2014. [Online]. Available: http://arxiv.org/abs/1404.7828

[34] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng, “TensorFlow: Large-scale machine learning on heterogeneous systems,” 2015, software available from tensorflow.org. [Online]. Available: http://tensorflow.org/

[35] D. Applegate, R. Bixby, V. Chvàtal, and W. J. Cook, “Concorde tsp solver,” 2006, software available from http://www.math.uwaterloo.ca. [Online]. Available: http://www.math.uwaterloo.ca/tsp/concorde/

[36] G. Casella and R. Berger, *Statistical Inference*. Duxbury Resource Center, June 2001.

[37] P. E. Hart, N. J. Nilsson, and B. Raphael, “A formal basis for the heuristic determination of minimum cost paths,” *IEEE Transactions on Systems Science and Cybernetics*, vol. 4, no. 2, pp. 100–107, July 1968.

[38] B. Efron and R. J. Tibshirani, *An introduction to the bootstrap*, ser. Mono. Stat. Appl. Probab. London: Chapman and Hall, 1993. [Online]. Available: https://cds.cern.ch/record/526679