Learning to Solve Travelling Salesman Problem with Hardness-Adaptive Curriculum

Zeyang Zhang, Ziwei Zhang, Xin Wang*, Wenwu Zhu*
Tsinghua University
zy-zhang20@mails.tsinghua.edu.cn, zwzhang@tsinghua.edu.cn
xin_wang@tsinghua.edu.cn, wwzhu@tsinghua.edu.cn

Abstract

Various neural network models have been proposed to tackle combinatorial optimization problems such as the travelling salesman problem (TSP). Existing learning-based TSP methods adopt a simple setting that the training and testing data are independent and identically distributed. However, the existing literature fails to solve TSP instances when training and testing data have different distributions. Concretely, we find that different training and testing distribution will result in more difficult TSP instances, i.e., the solution obtained by the model has a large gap from the optimal solution. To tackle this problem, in this work, we study learning-based TSP methods when training and testing data have different distributions using adaptive-hardness, i.e., how difficult a TSP instance can be for a solver. This problem is challenging because it is non-trivial to (1) define hardness measurement quantitatively; (2) efficiently and continuously generate sufficiently hard TSP instances upon model training; (3) fully utilize instances with different levels of hardness to learn a more powerful TSP solver. To solve these challenges, we first propose a principled hardness measurement to quantify the hardness of TSP instances. Then, we propose a hardness-adaptive generator to generate instances with different hardness. We further propose a curriculum learner fully utilizing these instances to train the TSP solver. Experiments show that our hardness-adaptive generator can generate instances ten times harder than the existing methods, and our proposed method achieves significant improvement over state-of-the-art models in terms of the optimality gap. The codes[1] are publicly available.

1 Introduction

The travelling salesman problem (TSP), as one important NP-hard problem, serves as a common benchmark for evaluating combinatorial optimization (CO) algorithms that have many practical real-world applications. As a trade-off between computational costs and solution qualities, a collection of approximate solvers and heuristics have been studied (Punnen, Margot, and Kabadi 2003; Helsgaun 2017). On the other hand, there is a recent advent of using machine learning to facilitate solving the NP-hard travelling salesman problem (TSP) in an end-to-end fashion (Vinyals, Fortunato, and Jaitly 2015; Kool, van Hoof, and Welling 2019; Khalil et al. 2017; Bello et al. 2017; Kwon et al. 2020; Deudon et al. 2018; Ma et al. 2019; Bresson and Laurent 2021). However, the existing methods independently sample training and testing data from the same distribution, i.e., the i.i.d. setting. More concretely, most methods directly adopt a uniform sampling within the unit square to generate TSP instances. Therefore, though these existing methods show reasonably good performance in such a setting, it is unclear whether the solver trained under the existing i.i.d. setting can actually solve the TSP problem by modeling and capturing the underlying relationships between instances and solutions or simply memorizing training instances. If the latter, the existing TSP solver will fail to handle when training and testing TSP have various distributions.

To answer this question, we first conduct some preliminary studies for the existing TSP solvers. Specifically, we replace the uniform samples in the testing phase with Gaussian mixture samples, which are significantly harder TSP instances whose solutions obtained from a TSP solver have a larger gap from the optimal solution (for more details, please refer to Section 3.2). The results are shown in Figure 1. The figure shows that indeed as we speculate, the existing TSP solvers fail to generalize to this more challenging setting where training and testing data have different distributions. As the distribution changes, the model performance degrades hundreds of times with respect to all metrics. The results clearly demonstrate that the existing TSP solvers have serious deficiencies when training and testing data have different distributions, producing more difficult instances.

To solve this problem, in this paper, we study learning-based TSP methods when training and testing data have different distributions using adaptive-hardness. However, there exist several challenges.

- Defining a quantitative hardness measurement is non-trivial since obtaining the optimal ground-truth solution of a TSP instance is NP-hard. Besides, as the TSP solver learns continuously during training, the hardness measurement must be updated adaptively.
- Even equipped with a hardness measurement, we need a

*Corresponding authors

[1]https://github.com/wondergo2017/TSP-HAC.

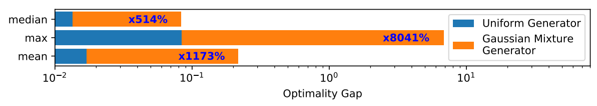


Figure 1: The optimality gap of an existing TSP solver trained on uniform samples and tested on Gaussian mixture samples. The maximum optimality gap grows 8041% times larger, showing the weakness of the existing models when training and testing data have different distributions.

generative model to generate TSP instances with different hardness levels. In particular, generating sufficiently difficult samples is challenging.

- After obtaining instances with desired hardness, we need to fully utilize these instances to train more powerful TSP solvers, especially for the ability of generalizing to different hardness.

To tackle these challenges, we first propose a principled hardness measurement to quantify the hardness of TSP instances. Specifically, we calculate the hardness as greedy self-improving potentials by comparing the current solver cost with a surrogate model. In this way, we avoid using the ground-truth optimal solution of TSP instances, calculating which is NP-hard and impractical. Besides, the hardness measurement is adaptive as the model learns continuously. Using the hardness measurement, we propose a hardness-adaptive generator, which can generate TSP instances with different hardness levels. Finally, we propose a curriculum learner to fully utilize the hardness-adaptive TSP instances generated by the generator. By learning instance weights, our method can train the TSP solvers more efficiently through curriculum learning.

We conduct extensive experiments to verify the designs of our proposed method. Experimental results show that our hardness-adaptive generator can produce instances 10x harder than the existing methods, i.e., fixed uniform samples. Besides, our proposed curriculum learner, together with the hardness-adaptive generator, can achieve significant improvement over state-of-the-art models in terms of the optimality gap when training and test data have different distributions.

In summary, we make the following contributions:

- We propose a principled hardness measurement using greedy self-improving potentials and surrogate models, avoiding the unbearable computational costs of calculating ground-truth optimal solution for TSP.

- We design a hardness-adaptive generator to efficiently and continuously generate instances with different levels of hardness tailored for model training.

- We propose a curriculum learner to fully utilize hardness-adaptive instances for more efficient model training with better performance.

2 Related Work

2.1 Travelling Salesman Problem

Recently, some works propose to use neural network models to facilitate solving TSP, including Pointer Networks (Bello et al. 2017; Nazari et al. 2018; Vinyals, Fortunato, and Jaitly 2015), Network Embedding (Khalil et al. 2017), and Graph Neural Networks (Joshi, Laurent, and Bresson 2019b; Kool, van Hoof, and Welling 2019; Joshi, Laurent, and Bresson 2019a; Ma et al. 2019; Fu, Qiu, and Zha 2020; Joshi et al. 2020; Bresson and Laurent 2021). There are two paradigms for training TSP models: supervised learning and reinforcement learning(RL) (Joshi, Laurent, and Bresson 2019b). Supervised methods adopt optimal routes as training labels, and the goal is to predict whether an edge exists in the optimal solution. Since ground-truth optimal solutions for TSP have to be used for training, this paradigm is not scalable for large-scale TSP instances. On the other hand, RL-based methods do not need optimal solutions of TSP. These methods commonly use a neural network encoder to obtain embeddings for each node, then use an RL controller to decide whether an edge is in the optimal solution step by step according to the state in RL. Once the output is obtained, the tour length acts as a negative reward for the RL controller to update the policy. These models obtain impressive performance when training and testing data have the same distribution, e.g., Kool et al. (Kool, van Hoof, and Welling 2019) achieves near perfect results for TSP instances with 50 nodes on uniformly sampled instances.

2.2 TSP Hardness and Size Generalization

Previous learning-based TSP methods generate training and testing data from the same distribution, e.g., uniform samples from a unit square. In such a setting, it is unclear whether the proposed method actually learns to solve TSP instances or simply memorizing the training data.

Recent works (Joshi et al. 2020; Fu, Qiu, and Zha 2020) post similar questions by showing that the learned TSP solvers fail to generalize to different TSP sizes (i.e., the number of nodes in TSP) in training and testing. Our work takes a step forward by showing that even for the same size, different distributions can lead to diverse difficulties and result in poor generalization.

There are few heuristic hardness measurements for TSP instances. For example, Smith et al. (Smith-Miles, van Hemert, and Lim 2010) define hardness-related instance features, which is closely related to the searching cost of heuristic solvers such as Lin–Kernighan (LK). Hemert (van Hemert 2005) defines the number of switching edges in LK as hardness. However, these measurements are based on non-learning-based TSP heuristics and are not suitable for learning-based solvers. In this paper, we define a principled hardness measurement for learning-based TSP solvers, which is also adaptive as the solver continuously learns.

2.3 Curriculum Learning

Curriculum learning (Portelas et al. 2020; Wang, Chen, and Zhu 2021) studies how to improve training strategies by manipulating the data distribution according to the model training stage, aiming to help the model train efficiently and obtain better performance (Bengio et al. 2009; Hacohen and Weinshall 2019).

A family of classical methods trains the target model with samples from easy to challenging (Bengio et al. 2009; Kumar, Packer, and Koller 2010; Platanios et al. 2019; Guo et al. 2018; Florensa et al. 2017), mimicking how humans learn. To perform curriculum learning, one has to first measure the hardness of samples and then use a curriculum scheduler to determine when and how to feed data into the model. In many fields, hardness measurement is based on domain knowledge (e.g., sentence length in machine translation (Platanios et al. 2019), signal to noise ratio in speech recognition (Ranjan and Hansen 2017), and others (Tudor Ionescu et al. 2016; Soviany et al. 2020; Wei et al. 2016)). For learning-based TSP, Lisicki et al. (Lisicki, Afkanpour, and Taylor 2020) takes the size of TSP as an indicator of hardness and train the model by gradually increasing the TSP size. In this paper, we focus on TSP instances of the same size but come from different distributions.

3 Problem Formulation and Preliminary Study

3.1 Notations and Problem Formulation

Following previous works, we focus on 2D-Euclidean TSP. Given the coordinate of n nodes, $\mathbf{X} = \{\mathbf{x}_i\}_{i=1}^n$, where $\mathbf{x}_i \in [0,1]^2$ is the 2-D coordinate of node i and the distance between node pairs $\mathcal{D}(\mathbf{x}_i, \mathbf{x}_j) = \|\mathbf{x}_i - \mathbf{x}_j\|_2$ is Euclidean, the objective is to find the shortest possible route that visits each node exactly once. The solution output by a TSP solver is a permutation of n nodes $\boldsymbol{\pi} = [\pi_1, \cdots, \pi_n]$ that minimizes the total tour length. Formally, the cost of solution $\boldsymbol{\pi}$ is defined as

$$\mathcal{C}(\boldsymbol{\pi}, \mathbf{X}) = \|\mathbf{x}_{\pi_n} - \mathbf{x}_{\pi_1}\|_2 + \sum_{i=1}^{n-1} \|\mathbf{x}_{\pi_i} - \mathbf{x}_{\pi_{i+1}}\|_2 . \quad (1)$$

Each instance $\mathbf{X}$ has a minimal (i.e., optimal) solution cost $\mathcal{C}^*(\mathbf{X}) = \min_{\boldsymbol{\pi}} \mathcal{C}(\boldsymbol{\pi}, \mathbf{X})$. Denote a TSP solver as M and the cost of its solution as $\mathcal{C}_M(\mathbf{X})$. Then $\mathcal{C}^*(\mathbf{X}) \leq \mathcal{C}_M(\mathbf{X}), \forall M$. Besides, no polynomial solver for TSP is known to date, i.e., an exact solver guaranteed to obtain the optimal solution for any TSP instance will evitably have an exponential time complexity, which is unbearable in practice. Learning-based TSP solvers aim to give near-optimal solutions while ensuring acceptable computational cost. We adopt the optimality gap as an optimization metric defined as:

$$\mathcal{G}_M(\mathbf{X}) = \frac{\mathcal{C}_M(\mathbf{X}) - \mathcal{C}^*(\mathbf{X})}{\mathcal{C}^*(\mathbf{X})}. \quad (2)$$

Note that we do not directly adopt $\mathcal{C}_M(\mathbf{X})$ or $\mathcal{C}_M(\mathbf{X}) - \mathcal{C}^*(\mathbf{X})$ because the optimal solution $\boldsymbol{\pi}^*$ remains the same when the distance of the instance is scaled, e.g., for $\mathbf{X}' = a\mathbf{X}$, where $a > 0$ is a constant. But in those cases, $\mathcal{C}_M(\mathbf{X}') = a\mathcal{C}_M(\mathbf{X})$ and thus is not comparable among instances with different scales. On the other hand, $\mathcal{G}_M(\mathbf{X}') = \mathcal{G}_M(\mathbf{X})$ and does not suffer from this problem.

For a dataset $\{\mathbf{X}_i\}_{i=1}^K$, where K is the number of instances, the optimality gap is averaged for all the instances, i.e., $\mathcal{G}_M(\{\mathbf{X}_i\}_{i=1}^K) = \frac{1}{K}\sum_{i=1}^K \mathcal{G}_M(\mathbf{X}_i)$, unless stated otherwise.

3.2 Preliminary Study

To investigate whether existing current learning-based TSP solvers can generalize to different distributions, we first report preliminary study results.

Gaussian Mixture Generator Following previous studies of TSP (Smith-Miles, van Hemert, and Lim 2010), Gaussian mixture generators can generate TSP instances with different hardness levels than the simple uniform sampling. Next, we introduce the Gaussian mixture generator in detail.

First, we sample the number of clusters n_c from a discrete uniform distribution

$$n_c \sim \mathcal{U}\{c_{\text{min}}, \cdots, c_{\text{max}}\}, \quad (3)$$

where c_{min} and c_{max} are hyper-parameters. In our experiments, we set $c_{\text{min}} = 3$ and $c_{\text{max}} = 7$. Denote the cluster that node i belongs to as c_i. Nodes have an equal probability of joining each cluster, i.e.,

$$c_i \sim \mathcal{U}\{1, \cdots, n_c\}. \quad (4)$$

Each cluster $i, 1 \leq i \leq n_c$, has a center vector $\boldsymbol{\mu}_i = (\mu_{i1}, \mu_{i2})$. The center vector is uniformly sampled in a square with length c_{DIST}, i.e.,

$$\boldsymbol{\mu}_i \sim \mathcal{U}\left[0, c_{\text{DIST}}\right]^2. \quad (5)$$

The coordinate of node i, $\mathbf{x}_i$, is sampled from a Gaussian distribution $\mathbf{x}_i \sim \mathcal{N}(\boldsymbol{\mu}_{c_i}, \mathbf{I})$. In this way, nodes belonging to the same cluster are close to each other. To normalize the ranges of TSP instances, we follow previous works and scale the node coordinates $\mathbf{X} = \{\mathbf{x}_i\}_{i=1}^K$ into a unit square by using a min-max projection function $\phi(\mathbf{x}_i; \mathbf{X})$ where $\phi(\cdot)$ is defined as:

$$\phi(\mathbf{x}_i; \mathbf{X}) = \frac{\mathbf{x}_i - \min(\mathbf{X})}{\max(\mathbf{X}) - \min(\mathbf{X})}, \quad (6)$$

where $\min$ and $\max$ are calculated dimension-wise.

3.3 Empirical Results

Next, we present our experimental results. Specifically, following previous works, the TSP solver is trained using uniform sampling (more details of the solver can be found in Section 5.1). Then, we test the solver using the Gaussian mixture generator.

The results when fixing the adjustable parameter c_{DIST} as 100 are shown in Figure 1. Though the existing solvers show reasonably good results when the testing distribution is identical to training (i.e., uniform sampling), the performance drops significantly when using Gaussian mixture as the testing distribution. In fact, the maximum optimality gap even scales by more than 80 times. These results clearly demonstrate the shortcoming of the existing models when training and testing data have different distributions.

Next, we show the results of varying c_{DIST} in Table 1. Empirically, we find that for the current learning-based TSP solvers, the optimality gap increases with c_{DIST}, i.e., instances with larger c_{DIST} are more difficult. Since uniform

Table 1: The experimental results when varying c_{DIST} for the Gaussian mixture generator.

c_{DIST}	10	20	30
Optimality Gap(%)	3.5 ± 0.2	7.2 ± 0.4	10.1 ± 0.6
c_{DIST}	50	70	100
Optimality Gap(%)	14.0 ± 1.0	17.1 ± 1.5	20.0 ± 1.6

sampling does not divide nodes into clusters, it is equivalent to $c_{\text{DIST}} = 0$ before projection function. Therefore, as c_{DIST} grows larger, the training and testing distribution become more diverse, and the weakness of the existing methods becomes more apparent.

4 Method

In this section, we introduce our proposed method. First, we introduce how to measure the hardness of TSP instances. Then, we propose a hardness-adaptive generator to sample hardness-diverse TSP instances. Lastly, we introduce a curriculum learning model to fully utilize these samples and better train the solver. The overall framework of our method is shown in Figure 2.

4.1 Hardness Measurment

To enable TSP solvers to handle various distributions, we need instances with different hardness levels. Therefore, we need a quantitative hardness measurement. Denote the hardness of instance $\mathbf{X}$ to solver M as $\mathcal{H}(\mathbf{X}, M)$. Notice that the hardness is solver-specific since the hardness can change as the solver is trained.

Some previous work (Lisicki, Afkanpour, and Taylor 2020) takes the TSP size n as an indicator of hardness since large-size instances are usually more complex and difficult to solve. In this paper, we take an orthogonal direction by considering the hardness of instances with the same size but have different distributions.

A straightforward idea is to use the optimality gap $\mathcal{G}_M(\mathbf{X})$ as the hardness since it naturally measures the quality of the solver compared to the optimal solution. However, it is impractical to obtain the optimal solution cost $\mathcal{C}^*(\mathbf{X})$ since itself is another NP-hard problem, i.e., the TSP decision problem. Therefore, we need to directly calculate $\mathcal{H}(\mathbf{X}, M)$ without calculating $\mathcal{C}^*(\mathbf{X})$.

Another challenge is that the solver M can vary greatly during training. For example, as the solver learns, instances considered hard in previous stages may not be hard anymore in the current stage, and it is crucial for the hardness measurement to capture such a drift. To solve this problem, we propose a hardness measurement using self-improving potentials

$$\mathcal{H}(\mathbf{X}, M) = \frac{\mathcal{C}_M(\mathbf{X}) - \mathcal{C}_{M'}(\mathbf{X})}{\mathcal{C}_{M'}(\mathbf{X})}, \tag{7}$$

where M' is a greedily updated surrogate model, e.g., when M is differentiable, we can conduct several steps of gradient descends to obtain M'. Intuitively, instead of comparing with the optimal solution, we compare M with a potential solution as the surrogate, i.e., if we can find M' with a much smaller cost than M, it means $\mathbf{X}$ is still "hard" for M. From another perspective, the hardness measurement is also a lower bound of the ground-truth optimality gap

$$\begin{aligned}\mathcal{H}(\mathbf{X}, M) &= \frac{\mathcal{C}_M(\mathbf{X}) - \mathcal{C}_{M'}(\mathbf{X})}{\mathcal{C}_{M'}(\mathbf{X})} \\ &\leq \frac{\mathcal{C}_M(\mathbf{X}) - \mathcal{C}^*(\mathbf{X})}{\mathcal{C}^*(\mathbf{X})} = \mathcal{G}_M(\mathbf{X}).\end{aligned} \tag{8}$$

The equality holds when our surrogate model M' produces the optimal solution for $\mathbf{X}$.

Notice that if solver M outputs routes with probabilities rather than being deterministic, we define cost function as an expectation, i.e., $\mathcal{C}_M(\mathbf{X}) = E_{p_M(\boldsymbol{\pi}|\mathbf{X})}\left[\mathcal{C}(\boldsymbol{\pi}, \mathbf{X})\right]$, where $p_M(\boldsymbol{\pi}|\mathbf{X})$ is the probability of solver M outputing route $\boldsymbol{\pi}$ for instance $\mathbf{X}$. We do not adopt minimum, i.e., $\mathcal{C}_M(\mathbf{X}) = \min_{\boldsymbol{\pi}}\left[\mathcal{C}(\boldsymbol{\pi}, \mathbf{X}), \boldsymbol{\pi} \in M\right]$ because such measurement is difficult to be optimized due to differentiability problems. Therefore, the hardness measurement is also defined as an expectation. As enumerating all possible routes is intractable, we sample routes to obtain an unbiased estimator.

4.2 Hardness-adaptive Generator

As shown in Section 3.2, naive i.i.d. sampling can not produce sufficiently difficult TSP instances. Next, we introduce a hardness-adaptive generator to sample hardness-diverse TSP instances using the hardness measurement $\mathcal{H}(\mathbf{X}, M)$.

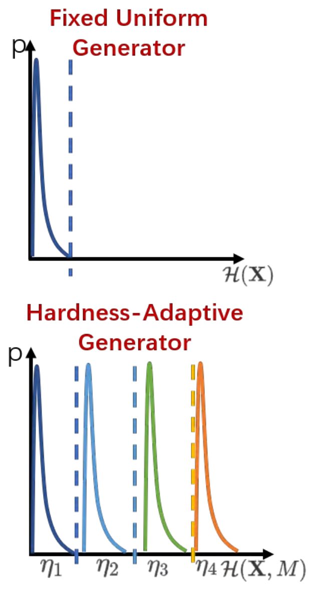
The main difficulty of designing a hardness-adaptive generator is to generate sufficiently difficult samples. To solve that challenge, inspired by energy-based models (Song and Kingma 2021), we adopt an energy function as the generative model. Specifically, we define the energy function using hardness as $E(\mathbf{X}|M) = -\mathcal{H}(\mathbf{X}, M)$. Then, the probability distribution is defined as

$$\begin{aligned}p(\mathbf{X}|M) &= \frac{\exp\left(-E(\mathbf{X})|M\right)}{\int_{\mathbf{X}'} \exp\left(-E(\mathbf{X}')|M\right) \mathrm{d}\mathbf{X}'} \\ &= \frac{\exp(\mathcal{H}(\mathbf{X}, M))}{\int_{\mathbf{X}'} \exp\left(\mathcal{H}(\mathbf{X}', M)\right) \mathrm{d}\mathbf{X}'}.\end{aligned} \tag{9}$$

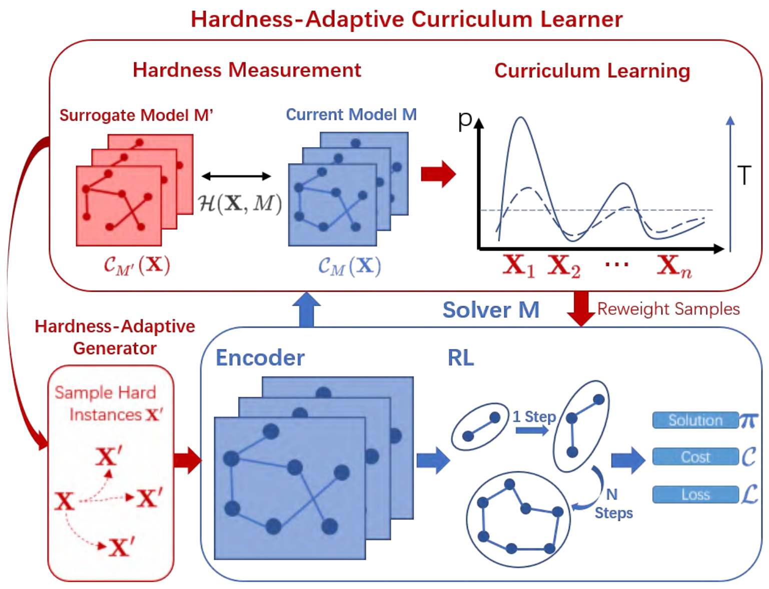
In other words, instances with larger $\mathcal{H}(\mathbf{X}, M)$ are more likely to be sampled. When generating the samples, we first randomly generate an instance $\mathbf{X}^{(0)}$, e.g., using uniform sampling. Then, inspired by Langevin Dynamics (Song and Ermon 2019), we further optimize the sample to increase the hardness by doing gradient ascend

$$\begin{aligned}\mathbf{X}^{(t)'} &= \mathbf{X}^{(t)} + \eta \nabla_{\mathbf{X}^{(t)}} \log p(\mathbf{X}^{(t)}) \\ &= \mathbf{X}^{(t)} + \eta \nabla_{\mathbf{X}^{(t)}} \mathcal{H}(\mathbf{X}^{(t)}, M), \\ \mathbf{X}^{(t+1)} &= \phi(\mathbf{X}^{(t)'}),\end{aligned} \tag{10}$$

where t denotes the number of optimization steps, $\phi(\cdot)$ is the projection function in Eq. (6) to ensure the validity of the generated samples, and η is the gradient step size. In the generator, we can adjust t and η to generate instances with different hardness.



(a) Generators


Hardness-Adaptive Curriculum Learner
Hardness Measurement
Curriculum Learning
Surrogate Model M'
Current Model M
$\mathcal{H}(\mathbf{X}, M)$
$\mathcal{C}_{M'}(\mathbf{X})$
$\mathcal{C}_{M}(\mathbf{X})$
p
T
$\mathbf{X}_1$ $\mathbf{X}_2$ ⋯ $\mathbf{X}_n$
Solver M
Reweight Samples
Hardness-Adaptive Generator
Sample Hard Instances $\mathbf{X}'$
$\mathbf{X}$
$\mathbf{X}'$
Encoder
RL
1 Step
N Steps
Solution π
Cost $\mathcal{C}$
Loss $\mathcal{L}$

(b) Our proposed hardness-adaptive curriculum learner

Figure 2: (a) A comparison between the fixed uniform generator and our proposed hardness-adaptive generator. (b) The curriculum learner framework: Solver M takes the generated samples and adopts an encoder and RL to calculate the cost. Information from the generator, samples, and the model is fed into the curriculum learner to exploit the current training stage, reweight samples, and update model parameters. The generator continues to generate hardness-adaptive samples for the current model to keep improving model performance.(Best viewed in color)

For methods outputting solutions with probabilities, directly calculating the gradient in Eq. (10) also needs to enumerate all possible solutions $\boldsymbol{\pi}$, which is intractable. Using policy gradient, we have

$$\begin{aligned}&\nabla_{\mathbf{X}} \mathcal{H}(\mathbf{X}, M)\\ &= E_{p_{M(\boldsymbol{\pi}\mid\mathbf{X})}}\left[\frac{\mathcal{C}_M(\mathbf{X})}{\mathcal{C}_{M'}(\mathbf{X})} \nabla_{\mathbf{X}} \log p_M(\boldsymbol{\pi} \mid \mathbf{X}) + \frac{\nabla_{\mathbf{X}} \mathcal{C}_M(\mathbf{X})}{\mathcal{C}_{M'}(\mathbf{X})}\right].\end{aligned} \tag{11}$$

In deriving the above equation, we assume $\mathcal{C}_{M'}(\mathbf{X})$ to be a constant. Then, we can sample routes from $p_M(\boldsymbol{\pi}|\mathbf{X})$ to obtain unbiased estimators of Eq. (11).

4.3 Hardness-adaptive Curriculum Learner

To learn a more powerful TSP solver, we explore hardness-adaptive samples through curriculum learning. In this paper, we adopt a simple reweighting-based curriculum strategy, but our method can be straightforwardly generalized to more advanced curriculum learning methods.

Consider a mixed dataset $\{\mathbf{X}_i\}_{i=1}^K$ with both hard and easy samples, e.g., generated by our hardness-adaptive generator and the fixed uniform generator. Intuitively, we should focus more on hard samples during training since they reflect the shortcomings of the current solver. Therefore, we propose to reweight TSP instance $\mathbf{X}_i$ using the hardness as follows,

$$w_i = \frac{\exp(\mathcal{F}(\mathcal{H}(\mathbf{X}_i, M))/T)}{\sum_j \exp(\mathcal{F}(\mathcal{H}(\mathbf{X}_j, M))/T)}, \tag{12}$$

Algorithm 1: Hardness-adaptive Curriculum Learner

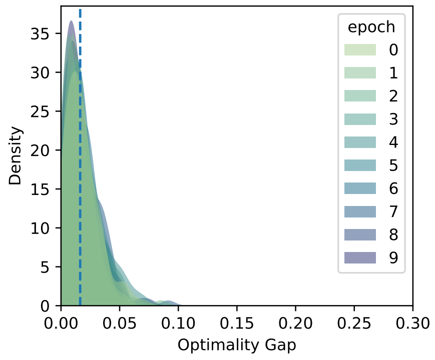
```
Require: Hardness-adaptive and uniform generator, TSP
    solver $M$, batch size $B$, training epochs $L$
 1: Initialize and warm up $M$ using uniform sampling
 2: for $l = 1, \dots, L$ do
 3:     Randomly generate dataset $D$ by uniform samples
        and hard samples using Eq. (10)
 4:     for $b = 1, \dots, |D|/B$ do
 5:         Get batch data $\{\mathbf{X}\}_{i=1}^B$ from $D$
 6:         Calculate hardness $\mathcal{H}(\mathbf{X}, M)$ for $\{\mathbf{X}\}_{i=1}^B$ using
            Eq. (7)
 7:         Calculate sample weights $w_i$ using Eq. (12)
 8:         Calculate gradients for each instance using
            Eq. (15)
 9:         Update model parameters using weighted gradients
            $\nabla \mathcal{L}_{\boldsymbol{\theta}}\left(\{\mathbf{X}_i\}_{i=1}^B\right) = \sum_i w_i \nabla \mathcal{L}_{\boldsymbol{\theta}}(\mathbf{X}_i)$
10:     end for
11:     Increase the curriculum learner temperature $T$
12: end for
```

where $\mathcal{F}(\cdot)$ is a transformation function and T is a temperature to control the stage of the curriculum. When the temperature is high, all the samples are treated roughly equally. As the temperature T goes down, the weight distribution shifts and harder samples are assigned larger weights than easy samples. For a data batch $\{\mathbf{X}_i\}_{i=1}^B$, the reweighted loss is calculated as follows

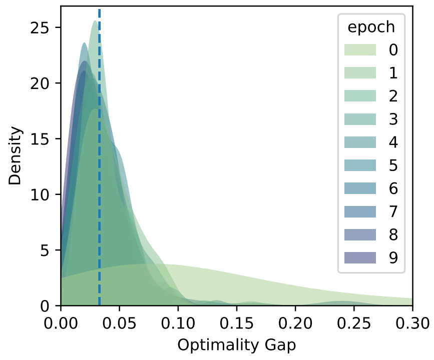
$$\mathcal{L}_{\boldsymbol{\theta}}\left(\{\mathbf{X}_i\}_{i=1}^B\right) = \sum\nolimits_i w_i \mathcal{L}_{\boldsymbol{\theta}}(\mathbf{X}_i), \tag{13}$$

Table 2: The results of different generators using a TSP solver pre-trained on uniform sampling

Testing Distribution	Model Cost	Optimal Cost	Optimality Gap (%)
Uniform Generator	5.80 ± 0.01	5.70 ± 0.01	1.70 ± 0.05
Hardness-adaptive Generator(η=5)	4.86 ± 0.05	4.35 ± 0.04	12.36 ± 1.21



(a) Samples generated by uniform generator



(b) Samples generated by our hardness-adaptive generator ($\eta = 5$).

Figure 3: The results of comparing the optimality gap of TSP instances generated by a uniform generator and our proposed hardness-adaptive generator. Our proposed generator can continuously generate instances with more diverse hardness levels. (Best viewed in color)

where $\boldsymbol{\theta}$ denotes the learnable parameters of the TSP solver. Following previous works (Kool, van Hoof, and Welling 2019), we use reinforcement learning to optimize the TSP solver. The loss function is defined as

$$\mathcal{L}_{\boldsymbol{\theta}}(\mathbf{X}) = E_{p_{M_{\boldsymbol{\theta}}}(\boldsymbol{\pi}|\mathbf{X})}\left[\mathcal{C}\left(\boldsymbol{\pi}|\mathbf{X}\right)\right]. \tag{14}$$

Then we can use policy gradient and REINFORCE (Williams 1992) algorithm to minimize the loss

$$\nabla\mathcal{L}_{\boldsymbol{\theta}}(\mathbf{X}) = E_{p_{M_{\boldsymbol{\theta}}}(\boldsymbol{\pi}|\mathbf{X})}\left[\left(\mathcal{C}(\boldsymbol{\pi}|\mathbf{X}) - \mathcal{C}_b(\mathbf{X})\right)\nabla\log p_{M_{\boldsymbol{\theta}}}(\boldsymbol{\pi} \mid \mathbf{X})\right], \tag{15}$$

where $\mathcal{C}_b(\mathbf{X})$ is the cost of a baseline to reduce gradient variances. Following (Kool, van Hoof, and Welling 2019), we set $\mathcal{C}_b(\mathbf{X})$ as a deterministic greedy rollout of the best model so far. The pseudo code is shown in Algorithm 1.

5 Experiments

In this section, we conduct experiments to answer the following questions:

- **Q1**: Can our hardness-adaptive generator continuously generate samples with diverse hardness levels than uniform sampling?
- **Q2**: Can our curriculum learning model train more powerful TSP solvers that are better generalizable to different distributions?

5.1 Experimental Setup

In our experiments, we focus on TSP-50, i.e., instances with 50 nodes. Throughout our experiments, we adopt GAT (Kool, van Hoof, and Welling 2019) as the encoder in our TSP solver with all hyperparameters are kept the same as in the original paper. Besides, we adopt Gurobi (Gurobi Optimization 2021), a dedicated non-learning-based TSP solver, to obtain the optimal cost $\mathcal{C}^*$ as ground-truths.

5.2 Experimental Details

For our experimental settings and model training, we largely follow (Kool, van Hoof, and Welling 2019) except that the TSP instances are generated using different methods. For experiments in Section 5.3, we sample 100,000 TSP instances for each training epoch from the uniform generator or our hardness-adaptive generator ($\eta = 5$). Then, we sample 100 randomly selected instances as the testing set and calculate the average optimality gap and standard deviation. For experiments in Section 5.4, we sample 100,000 instances for each training epoch. The testing dataset is composed of 10,000 instances generated by the Gaussian mixture generator. For the TSP solver, the GAT has three encoding layers with the embedding dimensionality 128 and the number of attention heads 8. Training methods such as batch normalization, tanh clipping of action logits and clipping of gradient norms are kept unchanged as in the original paper to stabilize the training process. We use Adam optimizer with the learning rate of 0.0001, the weight decay of 1.0, and the batch size of 512. The significance threshold used to update the RL baseline is $\alpha = 0.05$.

Table 3: Relationship between the optimality gap and η for our hardness-adaptive generator.

η	0.1	0.5	1	5	10
Optimality Gap (%)	0.7 ± 0.1	1.5 ± 0.2	2.9 ± 0.4	12.4 ± 1.2	14.7 ± 1.4

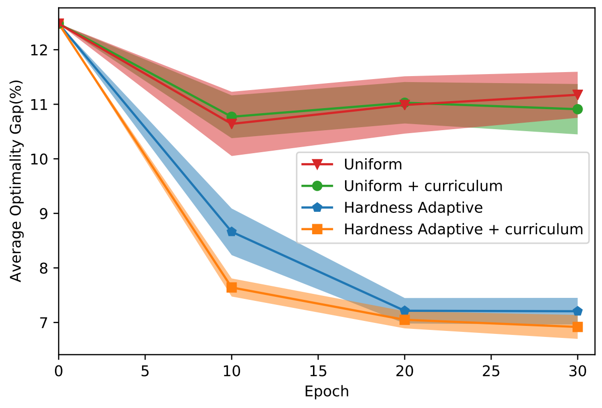


Figure 4: The experimental results when the training distribution is uniform and the testing distribution is Gaussian mixture (unknown to the model). Our proposed hardness-adaptive generator and curriculum learner significantly reduces the optimality gap and converges faster.

5.3 Generating Hardness-adaptive Instances

To answer **Q1**, we compare the hardness distribution of instances sampled by different generators. The TSP solver is pre-trained on samples from the uniform distribution.

First, we use a uniform generator and our proposed hardness-adaptive generator to generate instances and test the pre-trained solver. The results are shown in Table 2. Our proposed hardness-adaptive generator can generate more difficult TSP instances, demonstrating its effectiveness. These instances can be utilized to further improve our TSP solver.

Next, to test whether the generators can continuously generate hardness-adaptive samples as the solver learns, we fine-tune the solver using these generators, i.e., further optimize the solver using generators to generate training instances. Figure 3 shows the optimality gap (i.e., ground-truth hardness) distribution with different training epochs. As the epoch increases, samples generated by both generators tend to have a smaller optimality gap, but our proposed hardness-adaptive generator continuously generates instances with more diverse hardness levels.

Lastly, to verify the design of our hardness-adaptive generator that using a larger step size η can generate more difficult samples, we report the results of varying η in Table 3. As η grows larger, the optimality gap increases, indicating that the generated instance is more difficult. The results are consistent with our design.

5.4 Results when Training and Testing Distributions are Different

To answer **Q2**, we conduct experiments when training and testing TSP instances are from different distributions. Specifically, we consider two data generators for the training data as Section 5.3, i.e., a uniform generator and our hardness-adaptive generator. For hardness-adaptive generator, we set $\eta = 5$. Besides, we also compare two training paradigms: one using our proposed curriculum learner in Section 4.3 and another not using curriculum learning, i.e., all TSP instances have the same weights. For the testing data, we generate TSP instances using the Gaussian mixture generator introduced in Section 3.2. Notice that in all the scenarios, the testing distribution is unknown to the model. We repeat the experiments 15 times and calculate the average optimality gap evaluated on the testing dataset as the final results.

As shown in Figure 4, models trained using our hardness-adaptive generator significantly reduce the optimality gap, which demonstrates that our generated hardness-adaptive samples can greatly improve the TSP solver when training and testing data come from different distributions. Besides, the figure also shows that our proposed curriculum learner helps reduce variance and makes the training process faster.

6 Conclusion

In this paper, we explore whether the learning-based TSP solvers can generalize to different distributions. We propose a quantitative hardness measurement for TSP, a hardness-adaptive generator to generate instances with different hardness levels, and a curriculum learner to train the TSP solver. Experiments demonstrate the effectiveness of our proposed method in generating hardness-adaptive TSP instances and training more powerful TSP solvers when training and testing data have different distributions. For the future, it will deserve further investigations to extend our proposed hardness-adaptive curriculum learner to other combinatorial problems beyond the travelling salesman problem. We will also consider different hardness approximation methods, extention for non-differentiable solvers and the setting of continual learning in future works.

7 Acknowledgments

This work was supported in part by the National Key Research and Development Program of China No. 2020AAA0106300 and National Natural Science Foundation of China No. 62102222 and Tsinghua GuoQiang Research Center Grant 2020GQG1014. All opinions, findings, conclusions and recommendations in this paper are those of the authors and do not necessarily reflect the views of the funding agencies.

References

Bello, I.; Pham, H.; Le, Q. V.; Norouzi, M.; and Bengio, S. 2017. Neural Combinatorial Optimization with Reinforcement Learning. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Workshop Track Proceedings*. OpenReview.net.

Bengio, Y.; Louradour, J.; Collobert, R.; and Weston, J. 2009. Curriculum learning. In *Proceedings of the 26th annual international conference on machine learning*, 41–48.

Bresson, X.; and Laurent, T. 2021. The Transformer Network for the Traveling Salesman Problem. *CoRR*, abs/2103.03012.

Deudon, M.; Cournut, P.; Lacoste, A.; Adulyasak, Y.; and Rousseau, L. 2018. Learning Heuristics for the TSP by Policy Gradient. In van Hoeve, W. J., ed., *Integration of Constraint Programming, Artificial Intelligence, and Operations Research - 15th International Conference, CPAIOR 2018, Delft, The Netherlands, June 26-29, 2018, Proceedings*, volume 10848 of *Lecture Notes in Computer Science*, 170–181. Springer.

Florensa, C.; Held, D.; Wulfmeier, M.; Zhang, M.; and Abbeel, P. 2017. Reverse curriculum generation for reinforcement learning. *arXiv preprint arXiv:1707.05300*.

Fu, Z.; Qiu, K.; and Zha, H. 2020. Generalize a Small Pre-trained Model to Arbitrarily Large TSP Instances. *CoRR*, abs/2012.10658.

Guo, S.; Huang, W.; Zhang, H.; Zhuang, C.; Dong, D.; Scott, M. R.; and Huang, D. 2018. Curriculumnet: Weakly supervised learning from large-scale web images. In *Proceedings of the European Conference on Computer Vision (ECCV)*, 135–150.

Gurobi Optimization, L. 2021. Gurobi Optimizer Reference Manual.

Hacohen, G.; and Weinshall, D. 2019. On the power of curriculum learning in training deep networks. *arXiv preprint arXiv:1904.03626*.

Helsgaun, K. 2017. An extension of the Lin-Kernighan-Helsgaun TSP solver for constrained traveling salesman and vehicle routing problems. *Roskilde: Roskilde University*.

Joshi, C. K.; Cappart, Q.; Rousseau, L.; Laurent, T.; and Bresson, X. 2020. Learning TSP Requires Rethinking Generalization. *CoRR*, abs/2006.07054.

Joshi, C. K.; Laurent, T.; and Bresson, X. 2019a. An Efficient Graph Convolutional Network Technique for the Travelling Salesman Problem. *CoRR*, abs/1906.01227.

Joshi, C. K.; Laurent, T.; and Bresson, X. 2019b. On Learning Paradigms for the Travelling Salesman Problem. *CoRR*, abs/1910.07210.

Khalil, E. B.; Dai, H.; Zhang, Y.; Dilkina, B.; and Song, L. 2017. Learning Combinatorial Optimization Algorithms over Graphs. In Guyon, I.; von Luxburg, U.; Bengio, S.; Wallach, H. M.; Fergus, R.; Vishwanathan, S. V. N.; and Garnett, R., eds., *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA*, 6348–6358.

Kool, W.; van Hoof, H.; and Welling, M. 2019. Attention, Learn to Solve Routing Problems! In *ICLR*.

Kumar, M. P.; Packer, B.; and Koller, D. 2010. Self-paced learning for latent variable models. In *Advances in neural information processing systems*, 1189–1197.

Kwon, Y.; Choo, J.; Kim, B.; Yoon, I.; Gwon, Y.; and Min, S. 2020. POMO: Policy Optimization with Multiple Optima for Reinforcement Learning. In Larochelle, H.; Ranzato, M.; Hadsell, R.; Balcan, M.; and Lin, H., eds., *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*.

Lisicki, M.; Afkanpour, A.; and Taylor, G. W. 2020. Evaluating Curriculum Learning Strategies in Neural Combinatorial Optimization. *CoRR*, abs/2011.06188.

Ma, Q.; Ge, S.; He, D.; Thaker, D.; and Drori, I. 2019. Combinatorial Optimization by Graph Pointer Networks and Hierarchical Reinforcement Learning. *CoRR*, abs/1911.04936.

Nazari, M.; Oroojlooy, A.; Snyder, L. V.; and Takác, M. 2018. Reinforcement Learning for Solving the Vehicle Routing Problem. In Bengio, S.; Wallach, H. M.; Larochelle, H.; Grauman, K.; Cesa-Bianchi, N.; and Garnett, R., eds., *Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, December 3-8, 2018, Montréal, Canada*, 9861–9871.

Platanios, E. A.; Stretcu, O.; Neubig, G.; Poczos, B.; and Mitchell, T. M. 2019. Competence-based curriculum learning for neural machine translation. *arXiv preprint arXiv:1903.09848*.

Portelas, R.; Colas, C.; Weng, L.; Hofmann, K.; and Oudeyer, P. 2020. Automatic Curriculum Learning For Deep RL: A Short Survey. In Bessiere, C., ed., *Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence, IJCAI 2020*, 4819–4825. ijcai.org.

Punnen, A. P.; Margot, F.; and Kabadi, S. N. 2003. TSP Heuristics: Domination Analysis and Complexity. *Algorithmica*, 35(2): 111–127.

Ranjan, S.; and Hansen, J. H. 2017. Curriculum learning based approaches for noise robust speaker recognition. *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, 26(1): 197–210.

Smith-Miles, K.; van Hemert, J. I.; and Lim, X. Y. 2010. Understanding TSP Difficulty by Learning from Evolved Instances. In Blum, C.; and Battiti, R., eds., *Learning and Intelligent Optimization, 4th International Conference, LION 4, Venice, Italy, January 18-22, 2010. Selected Papers*, volume 6073 of *Lecture Notes in Computer Science*, 266–280. Springer.

Song, Y.; and Ermon, S. 2019. Generative Modeling by Estimating Gradients of the Data Distribution. *CoRR*, abs/1907.05600.

Song, Y.; and Kingma, D. P. 2021. How to Train Your Energy-Based Models. *CoRR*, abs/2101.03288.

Soviany, P.; Ardei, C.; Ionescu, R. T.; and Leordeanu, M. 2020. Image difficulty curriculum for generative adversarial networks (CuGAN). In *The IEEE Winter Conference on Applications of Computer Vision*, 3463–3472.

Tudor Ionescu, R.; Alexe, B.; Leordeanu, M.; Popescu, M.; Papadopoulos, D. P.; and Ferrari, V. 2016. How hard can it be? Estimating the difficulty of visual search in an image. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2157–2166.

van Hemert, J. I. 2005. Property Analysis of Symmetric Travelling Salesman Problem Instances Acquired Through Evolution. In Raidl, G. R.; and Gottlieb, J., eds., *Evolutionary Computation in Combinatorial Optimization, 5th European Conference, EvoCOP 2005, Lausanne, Switzerland, March 30 - April 1, 2005, Proceedings*, volume 3448 of *Lecture Notes in Computer Science*, 122–131. Springer.

Vinyals, O.; Fortunato, M.; and Jaitly, N. 2015. Pointer Networks. In Cortes, C.; Lawrence, N. D.; Lee, D. D.; Sugiyama, M.; and Garnett, R., eds., *Advances in Neural Information Processing Systems 28: Annual Conference on Neural Information Processing Systems 2015, December 7-12, 2015, Montreal, Quebec, Canada*, 2692–2700.

Wang, X.; Chen, Y.; and Zhu, W. 2021. A Survey on Curriculum Learning. *IEEE Transactions on Pattern Analysis and Machine Intelligence*.

Wei, Y.; Liang, X.; Chen, Y.; Shen, X.; Cheng, M.-M.; Feng, J.; Zhao, Y.; and Yan, S. 2016. Stc: A simple to complex framework for weakly-supervised semantic segmentation. *IEEE transactions on pattern analysis and machine intelligence*, 39(11): 2314–2320.

Williams, R. J. 1992. Simple Statistical Gradient-Following Algorithms for Connectionist Reinforcement Learning. *Mach. Learn.*, 8: 229–256.