Ask a Simpler Question, Get a Smarter Answer: Curriculum Learning for Solving the Traveling Salesman Problem

Spencer Compton[*1] **Tong Zhao**[*1]

Abstract

Algorithms take time. In general, algorithms for combinatorial optimization problems are typically designed by knowledgeable experts. Many problems (e.g. NP-Hard problems) require the design of approximation algorithms or heuristics. Not only is this time-intensive, but different heuristics work well in different scenarios. Recent work such as (Drori et al., 2020) uses reinforcement learning to learn approximations and avoid the need for hand-designed heuristics. In the traveling salesman problem (TSP), one must design a schedule of minimal length that visits all points in a given set. Work such as (Nazari et al., 2018) has proposed architectures that use reinforcement learning to learn high-quality TSP and vehicle routing problem (VRP) policies. We build upon the results of such work by examining a training methodology based on that of curriculum learning. Similar to how human experts typically approach algorithm design, our approach enables the learning algorithms to first gain insights from simpler versions of a problem (e.g. TSP when all points are on a line) before tackling the general problem. We experimentally evaluate our methodology with the architecture of (Nazari et al., 2018) on both human-designed curriculums and automated curriculum learning. We discuss how our approach is agnostic to the underlying model's architecture and its applications to the few-shot learning or distribution shift settings.

[*]Equal contribution [1]Massachusetts Institute of Technology, Cambridge, Massachusetts, USA. Correspondence to: Spencer Compton <scompton@mit.edu>, Tong Zhao <tzhao@mit.edu>.

Proceedings of the 37^{th} International Conference on Machine Learning, Online, PMLR 119, 2020.

1. Introduction

1.1. The Problem

We tackle the problem of extending learning methods such as (Drori et al., 2020) to better solve combinatorial optimization tasks such as the traveling salesman problem (TSP) and vehicle routing problem (VRP). In TSP, we optimize a tour of minimum length that visits all points in a graph. While there are many variants, VRP is often viewed as a generalization of TSP where we design schedules for some set of vehicles such that all points are serviced by a vehicle. The variant of VRP studied by (Drori et al., 2020) does this with M vehicles, and minimizes the longest tour length of any single route such that every node is visited by at least one vehicle. Extensive study of these problems from many perspectives are motivated by their relevance to real-world problems. For example, a package delivery company determining an efficient schedule for a truck to delivery all its assigned packages is an instance of TSP. Likewise, such a company determining a schedule for all its trucks to delivery all its packages (where any truck can hold any package) is solving an instance of VRP. The VRP problem is of particular interest because of the prevalence of Mobility-on-Demand ridesharing (e.g. Uber, Lyft). Technical hurdles for this problem are that operating regions (e.g. cities) likely have vastly different distributions of rider requests and vehicles. We also note that distributions within operating regions can vastly shift over time (e.g. seasonal patterns, sudden changes like COVID-19). Automated solutions that learn how to solve such problems are desirable not just because they remove the need for a human expert but also because they can learn structure specific to the distribution they train on (e.g. patterns for a particular city's passenger demands). While much work has been done in designing architectures for learning such solutions, less focus has been placed on training methods that improve performance in general and in settings such as few-shot learning (e.g. only a few samples from a particular city) or distribution shift (e.g. sudden changes like COVID-19). Improvement in these (realistic and common) settings is crucial, as learned solutions must be reliable to deserve the stakeholders' trust.

We focus on the problem of designing training techniques for these combinatorial optimization problems that lever-

age existing learning architectures. In concrete terms, our primary goal is to incorporate meta-learning techniques to improve the performance of existing architectures both in the general setting and in the few-shot learning and distribution shift settings. Our guiding philosophy is that immediately training on the desired test distribution is too ambitious to achieve best results. When human experts design algorithms, they typically consider simpler special-cases before attempting the general problem. This enables the human to confirm their understanding of the problem's workings and identify patterns that may lead towards crucial insights. We aim to provide the same opportunities to learning algorithms with our training method that leverages curriculum learning techniques.

In Section 1.2, we discuss previous work in the context of learning for combinatorial optimization and curriculum learning. In Section 2, we detail our training methodology for curriculum-based learning, subtask discovery, and RL-based curriculum learning. In Section 3, we provide experimental results (for completed experiments) and experimental design (for incomplete experiments) to evaluate the effectiveness of our methods. Finally, in Section 4, we discuss the implications of our work and potential future work.

1.2. Previous Work

Classical algorithms and heuristics. Both TSP and VRP have been proven to be NP-complete (Lenstra & Kan, 1981). However, it is still possible to design approximation algorithms with solutions that are provably close to an optimal solution. (Christofides, 1976) proved a classical result that there exists an efficient 3/2-approximation algorithm for TSP (meaning the size of this algorithm's tour is at most 50% more than the optimal). While this approximation ratio was the best result for about 40 years, the very recent work of (Karlin et al., 2020) improved this approximation ratio to about $(3/2 - 10^{-36})$ (a very small improvement, but it shows a monumental result that the longstanding bound of 3/2 is not tight and will likely be further improved). But in practice, heuristics without provable approximation ratios are often used as they perform better and leverage structure typically present in real-world problem instances. Often referred to as "solvers", many for TSP such as Google OR-Tools (Perron & Furnon), Concorde (Applegate et al., 2006), and Gurobi (Gurobi Optimization, 2020), are used extensively and have had their performance analysed thoroughly (McMenemy et al., 2019). Both provable approximations and heuristic solvers require extensive time for experts to hand-design. Moreover, they typically cannot adapt to new patterns within a certain use-case. Thus, it is appealing to utilize machine learning techniques and enable the automatic creation of such algorithms (that would ideally even out-perform hand-designed solutions).

Learning solutions for combinatorial optimization. Many recent works utilize machine learning techniques to tackle TSP and VRP variants (Zhang & Yang, 2020; Falkner & Schmidt-Thieme, 2020; Peng et al., 2019; Gao et al., 2020). (Nazari et al., 2018) is known for providing a machine learning solution that outperformed traditional methods such as Google OR-Tools. (Vesselinova et al., 2020) provides a more general survey of combinatorial optimization on graphs. (Drori et al., 2020) proposes a framework to learning approximation algorithms for graph optimization problems.

We concern ourselves with improving the training methods for these architectures. Primarily, we focus on utilizing learning from multiple tasks to help improve solution quality in general, and in the few-shot and distribution shift setting. A few recent papers approach multiple tasks and generalization that we hope to build upon. (Sun et al., 2020) develops a machine learning solution to TSP and examines good generalization ability to instances of different sizes or characterizations than it was trained on, but does use instances from one distribution to help train for solving instances of another distribution. do much to purposefully improve generalization. (Joshi et al., 2020) launches a deep investigation into generalization of machine learning solutions for TSP. In their work, they focus on generalization in terms of sizes of the problem instance, showing a capability to learn solutions for problem instances larger than any trained on. (Osaba et al., 2020) explores Evolutionary Multitasking in the context of VRP and found positive genetic transfer between tasks from different datasets. This provided an intuition for our work that if positive genetic transfer occurs between datasets, learning solutions to one task can almost certainly help learn another. From this intuition, we propose a training method that intentionally exploits the benefit of learning some tasks before others to produce stronger results.

To accomplish this, we utilize curriculum learning techniques. We note that the concurrent work of (Lisicki et al., 2020) was published during the course of our work. (Lisicki et al., 2020) effectively uses curriculum learning in their training method to train models that perform very well on many different different sizes simultaneously (with the goal being to do almost as well on all sizes as if they had trained on just that size). Our work takes a different perspective, using curriculum learning to train on instances of different characterizations to perform even better on a particular test distribution than if we had trained directly on that distribution.

Curriculum learning. (Bengio et al., 2009) popularly introduced the concept of curriculum learning to the field. In their work, they evaluated experiments that showed neural networks training significantly better on a (human-designed)

curriculum (i.e. an ordering of the training data) than if they trained in a random order. The underlying philosophy was that humans learn on increasingly difficult concepts and that it likely helps such models as well. The work of (Graves et al., 2017; Matiisen et al., 2019) show strong performance for automatic curriculum learning. For example, (Graves et al., 2017) uses an RL-based *teacher* to decide what the *student* will train on each epoch. The teacher learned how to best help the student learn through the use of *progress signals* that are heuristics meant to serve as a proxy for the student's progress from that epoch. Thus, every epoch the teacher chooses a task for the student to train on as its action, and receives the student's progress signal as a reward. We leverage this architecture our work and make use of their evaluation of what progress signals work best.

2. Methods

2.1. Baseline Learning Architecture

We note that the focus of our work is on training methods and not the underlying architecture. Moreover, our approach is agnostic to said architecture. As such, we will not detail particulars of any architecture for learning solutions to such combinatorial optimization problems. For simplicity, throughout the work (particularly in the context of experiments discussed in Section 3) one can assume the underlying baseline architecture is that of (Nazari et al., 2018).

From now on, we completely abstract this baseline learning architecture and refer to it as the *student*. The standard method for training the student is to train it on many epochs of tasks from the same distribution as the test distribution. We now detail alternative methods for training the student.

2.2. Curriculum-Based Training

We train the student using key ideas from curriculum learning. For example, suppose we want to train a model to perform well on TSP instances where the points are sampled uniformly from a unit square. Standard training methods such as those used by (Nazari et al., 2018) would simply train on TSP instances from the same distribution. Motivated by curriculum learning, we consider that it may be beneficial to train on what we call *pedagogical distributions*, followed by the actual test distribution. The key motivation is that human experts in algorithm design typically consider simpler variants (e.g. special cases) before trying to solve the general problem. The benefit of doing so for humans is that they confirm their understanding of the problem and potentially observe patterns that generalize to help solving the general problem. Pedagogical distributions serve this purpose in our training process. We train the student on a pedagogical distribution (or multiple) which are easier to learn/solve for some number of epochs, and then finally train on the test distribution utilizing knowledge learned from training on the pedagogical distributions.

For example, consider training on a pedagogical distribution for a TSP task where all points are sampled uniformly from the border of a diamond shape given in Fig. 1. Solving instances from this distribution is fairly simple, as it is approximately optimal to simply traverse points in clockwise or counterclockwise order. Then, we would train on the test distribution of the unit square when the model has already learned how to solve the simpler pedagogical distribution first.

More systematically, every epoch the student interfaces with the teacher. Here, the *teacher* tells the student what distribution to train on for the next epoch. Afterwards, the teacher may query information regarding the student to best determine what distribution the student should train on. We examine both the setting where the teacher's decisions are hard-coded (i.e. a hand-designed curriculum) or are learned by an agent. As this methodology is enough to support hand-designed curriculum learning, we further our methodology for automated curriculum learning in this setting.

2.3. Automated Pedagogical Distribution Discovery

In terms of automated curriculum learning for such combinatorial optimization, we consider two dimensions of automation. One dimension is selecting training data for the student each epoch, where the set of pedagogical distributions may be fixed or hand-designed. The other dimension is the discovery of pedagogical distributions to help in the training process. Even without automatic pedagogical distribution discovery, curriculum learning is very helpful in this setting for removing the need for experts while potentially improving performance. For example, many humans who aren't experts in algorithm design could design special-cases to be pedagogical distributions.

However, it is clearly desirable to remove dependence on humans from the system when possible. Previous work has found success in automated curriculum learning when tasks can be parameterized, which enables a teacher to search over the parameterization for tasks on which it is helpful for the student to train. For example, (Lee et al., 2020) uses this technique to learn quadrupedal locomotion by parameterizing the terrains the agent would need to traverse. As such, we provide a simple parameterization for Euclidean TSP and VRP tasks. To accomplish, we divide the unit square into T rows and columns, effectively creating T^2 square *tiles* of equal size. With this, we denote every task in this parameterization with a $T \times T$ square matrix P, where $P_{i,j}$ denotes the probability that a randomly sampled point from this distribution comes from the tile corresponding to the i-th row and j-th column (all entries of P are non-negative and sum to 1). Within a tile, every point is sampled with

uniform probability. As T increases in size, this parameterization gets arbitrarily close to being able to approximate any distribution over the unit square.

Unfortunately, searching over the parameterization for a helpful pedagogical distribution is more challenging in our setting than, e.g., the aforementioned locomotion work of (Lee et al., 2020). In the locomotion work, their goal is to have very small loss (i.e. correctly traverse) for every task (i.e. terrain). This is not the case for our setting. For example, our model should have a much shorter tour for a TSP task where two points are right next to each other, compared to a task where two point are at opposite corners of the unit square. More concretely we desire low *optimality gap*, the percentage increase of the learned solution compared to an optimal solution. As one needs to compute the optimal solution to know the optimality gap of a proposed solution, it is time-consuming to compute and not easily differentiable. Thus, we utilize black-box function optimization methods such as (Costa & Nannicini, 2018). To find a helpful pedagogical distribution, we fix the curriculum schedule and search for the best pedagogical distribution (e.g. train on distribution P for 10 epochs and then the target distribution for 20 epochs). This means we use black-box optimization to minimize the function $f(P)$, where $f(P)$ is the loss of a model after training it using the fixed curriculum schedule and P as the pedagogical distribution. Large values of T enable more options for P, but smaller T likely help the black-box optimization subroutine approach an optima more easily.

This provides a method for the dimension of automation focusing on pedagogical distribution discovery. Now, we focus on the dimension corresponding to learning how to select a distribution for the student to train on each epoch.

2.4. RL-based Automated Curriculum Learning

Given a fixed set of pedagogical distributions, how does the teacher learn what distributions to have the student train with? We utilize the automated curriculum framework presented in (Graves et al., 2017). Every epoch, the RL-based teacher takes an action corresponding to what task they have the student train on. Ideally, the teacher would have a reward corresponding to how much the student progressed towards proficiency in the test distribution. However, this is not very measurable (one can have progress that does not reflect in directly evaluating the model on the test distribution). For example, learning how to collect wood is progress towards being able to make a ladder but it does not immediately improve one's ladder-making abilities. This led (Graves et al., 2017) to use *progress signals* that act as proxies for such progress. In their work, they evaluated many progress signals and found that prediction gain (PG) performed best. (Bellemare et al., 2016) introduced prediction gain, which is the difference between the loss of the model on the training samples before and after training. More formally, let θ denote the parameters of the model before training this epoch, θ' denote the parameters of the model after training this epoch, and x denote the training samples for this epoch. Prediction gain is then equal to $L(x, \theta) - L(x, \theta')$. The intuition here is that using a prediction gain progress signal encourages the model to train on tasks where it "learns" the most in general. This works well with the previously mentioned ladder example, as learning how to collect wood by training on said task would give positive reward. Thus, a valid (and promising) method for automated curriculum learning in our setting would be having an RL-based teacher with its action space being exactly the set of pre-determined pedagogical distributions and the reward being the prediction gain progress signal. In Section 3.5 we also outline this progress signals potential advantages for the settings of few-shot learning and distribution shift.

We use this method, but also propose a different progress signal. Note that a teacher using prediction gain as a reward will act entirely independently of what the test distribution is. This makes sense in the context of (Graves et al., 2017), as they place a larger emphasis on the setting of training a model that does well in many tasks simultaneously. For our setting where we have a particular test distribution, we believe it makes sense to provide rewards that motivate the teacher to train the student in a manner that ultimately helps performance on the testing distribution as much as possible. (Graves et al., 2017) evaluated target prediction gain (TPG) which is the difference in loss on the test distribution before and after training that epoch. More formally, let x' denote samples from the test distribution, target prediction gain is $L(x', \theta) - L(x', \theta')$. This performed worse than prediction gain both in their evaluation and according to the philosophy of the aforementioned ladder example. We combine these progress signals to produce the novel test-aware prediction gain (TAPG) which is a weighted sum of PG and TPG, or $(L(x, \theta) - L(x, \theta') + c(L(x', \theta) - L(x, \theta)))$ for some c. For simplicity we can consider $c = 1$, but note that performance would likely improve with hyperparameter optimization over c. This progress signal enables our teacher to earn rewards both from learning in general and for emphasizing learning solutions for the test distribution well.

We additionally consider the setting where pedagogical tasks are not pre-determined (either by a human or methods proposed in Section 2.3). In this setting, we simply replace the teacher's action space with the set of all valid parameterizations (as discussed in Section 2.3) P. Of course, this massive increase in action space potentially makes convergence towards high-quality teacher policies take much longer (if ever obtained).

Comparison to SuperLoss method. The recent work of

$$\begin{bmatrix} 0 & 0 & 0 & 0.0625 & 0.0625 & 0 & 0 & 0 \\ 0 & 0 & 0.0625 & 0 & 0 & 0.0625 & 0 & 0 \\ 0 & 0.0625 & 0 & 0 & 0 & 0 & 0.0625 & 0 \\ 0.0625 & 0 & 0 & 0 & 0 & 0 & 0 & 0.0625 \\ 0.0625 & 0 & 0 & 0 & 0 & 0 & 0 & 0.0625 \\ 0 & 0.0625 & 0 & 0 & 0 & 0 & 0.0625 & 0 \\ 0 & 0 & 0.0625 & 0 & 0 & 0.0625 & 0 & 0 \\ 0 & 0 & 0 & 0.0625 & 0.0625 & 0 & 0 & 0 \end{bmatrix} \Longrightarrow$$

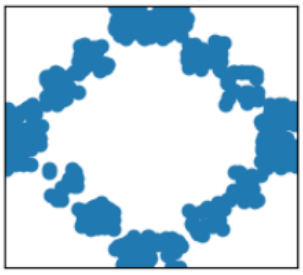

Figure 1. Example of $n = 200$ nodes sampled from a distribution corresponding to P.

(Castells et al., 2020) introduced SuperLoss, a universal approach for automated curriculum learning. The key idea behind SuperLoss is to add a loss function over the original loss function. This new loss function acts to make examples with high loss be weighted lower than examples with low loss. Philosophically, this decreases the importance of hard samples and increases the importance of easy samples. This is a key tenet of curriculum learning, as the goal is to train on (presumably easier) samples that one is more capable of learning from before (presumably harder) samples that they are not yet equipped to learn from. We note strong performance of this approach in their evaluation and ease of implementation. A comparison of our proposed methods based on (Graves et al., 2017) and usage of SuperLoss would likely be very valuable. That said, we provide discussion for why we believe our proposed methods are more fitting in the setting we study. In our setting, we seek to optimize performance of a test distribution that does not directly correspond to the distribution of tasks available to train on. Consider, for example, training on a set of pedagogical distributions that correspond to data from various cities or hand-made special-case distributions. The approach of SuperLoss does not take into account what the test distribution is. In particular, it would essentially be training on all data from easiest to hardest. This does not consider the *dimension of relevance* to the test distribution. For example, it does not work to place a higher emphasis or train more epochs based on how relevant a task is to the test distribution (only based on global difficulty of a task). If there are cities or hand-made distributions that are not very relevant to the test distribution, we believe using SuperLoss may prevent convergence to as high-quality of a solution or make such convergence take longer (as it would need to spend time training on irrelevant data). We believe it is important for such learning methods to be robust to poor curation of pedagogical tasks where many (potentially the majority) are not relevant to the test task. To emphasize our rationale, consider an analogous setting for an agent learning an advanced computer science course. If the supplied training tasks corresponded to courses from various disciplines and difficulties, SuperLoss would effectively have the model train on the courses in increasing order of difficulty (independent of their discipline). However, training on history or biology courses of any difficulty would likely not be very helpful. In comparison, our proposed approach uses the proxy of progress signals to learn a policy that both tries to teach easier tasks but also those that are relevant (likely focusing on computer science courses in increasing order of difficulty). This rationale leads us to believe that our proposed methods are valuable in this setting where our test distribution does not correspond to the training distributions. However, we still believe that an empirical comparison of both methods would be valuable.

3. Results

3.1. Hand-Designed Curriculum-Based Learning

We evaluated the performance of nine hand-designed curricula against the performance of training on the uniform distribution. We run an implementation of (Nazari et al., 2018) based off of the implementation by Matthew Veres (mveres01)[1]. The test distribution for all experiments was uniform over the unit square. This decision was made because (Nazari et al., 2018) focuses its distribution on the uniform distribution and our goal was to show a fair comparison of our training methods with the classical training method they used. This experiment design mirrors that of the original curriculum learning paper of (Bengio et al., 2009), that showed custom curriculums had the potential to significantly improve training in neural networks. Our goal is to analogously show the performance of intuitive, hand-designed curriculums in combinatorial optimization. We depict our hand-designed pedagogical distributions in Fig. 2.

We detail the performance of each of these curriculums compared to Google OR-Tools and training only on uniform distribution in Table 2. We describe each of the nine hand-designed curriculums in Table 1. For each of the learning-based training methods, we trained for a total of 30

[1]Code available at `github.com/tzhao42/curriculum-tsp`

epochs. We modify the implementation of Matthew Veres to generate a new set of training instances of uniform samples each epoch (previously they used the same for all epochs) as this improves performance and we desire a fair comparison. Google OR-Tools was run with a 10s timeout (for sake of computation time) to estimate the optimal tour. The optimality gap was estimated by taking the average tour length of a method and dividing that by the average tour length obtained by Google OR-Tools. In future iterations, we intend on calculating optimality gap more precisely (and as the average optimality gap between the optimal solution of a specific instance and the obtained solution for that specific instance). Our results in Table 2 indicate some curriculums performed better than training only on uniform for TSP-20, and no curriculums performed better than training only on uniform for TSP-50. However, our loss plots in Fig. 3 indicate that these results may not have fully converged (multiple methods are underfitting) and we must re-evaluate these experiments.

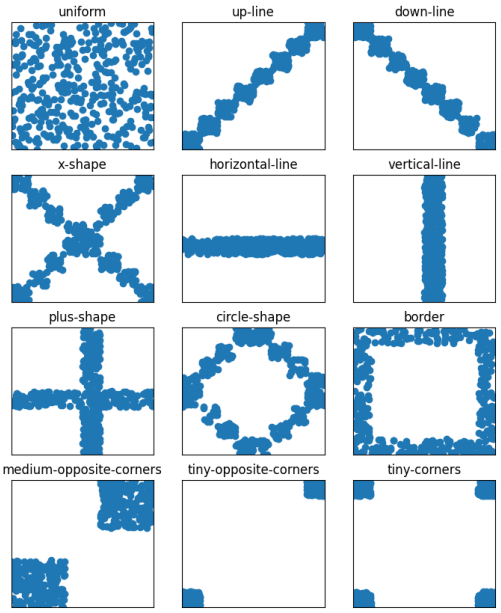

Figure 2. Different subtask distributions used in training. The majority of the curricula used consisted of 10 epochs of one subtask distribution, and then 20 epochs of the uniform distribution. Note that all these subtasks were human designed before any training or examination of results.

Curriculum	Details
Uniform	30 epochs uniform
1	6 epochs tiny-opposite-corners 6 epochs medium-opposite-corners 18 epochs uniform
2	4 epochs tiny-opposite-corners 4 epochs tiny-corners 4 epochs border 18 epochs uniform
3	4 epochs tiny-opposite-corners 4 epochs tiny-corners 4 epochs circle-shape 18 epochs uniform
4	2 epochs horizontal-line 2 epochs vertical-line 2 epochs plus-shape 2 epochs down-line 2 epochs up-line 2 epochs x-shape 18 epochs uniform
5	4 epochs down-line 4 epochs up-line 4 epochs circle 18 epochs uniform
6	10 epochs circle-shape 20 epochs uniform
7	10 epochs tiny-corners 20 epochs uniform
8	10 epochs border 20 epochs uniform
9	10 epochs down-line 20 epochs uniform

Table 1. Different curricula used for training. After each epoch, the dataset is regenerated according to the current distribution – this mitigates the effects of curricula introducing a larger number of unique problem instances. Note that all curricula have a total of 30 epochs.

3.2. Learning Pedagogical Distributions

We plan to evaluate the effectiveness of the proposed method of automatic pedagogical distribution discovery from Section 2.3. We will configure a few $f(P)$ based on a different fixed curriculum schedules (e.g. 10 epochs on P then 30 epochs on uniform) and evaluate the effectiveness of black-box optimization (specifically, the work of (Costa & Nannicini, 2018)) of finding a pedagogical distribution P such that P is helpful in training. As a control, we will compare this to both only training on uniform for the same number of epochs and training on a number of random P (sampled from uniform simplex) equal to the number of evaluations of our black-box optimization methods. This would show not only if our proposed method can find peda-

Task	Curriculum	Avg Tour	Est Opt Gap (%)
Tsp-20	GOR-tools	3.8218	-
	Uniform	4.0003	4.670
	1	4.0260	5.343
	2	4.0046	4.783
	3	3.9924	4.463
	4	4.0108	4.945
	5	3.9935	4.492
	6	3.9922	4.458
	7	4.0049	4.790
	8	4.0096	4.913
	9	4.0306	5.463
Tsp-100	GOR-tools	8.2835	-
	Uniform	8.5963	3.776
	1	9.0801	9.616
	2	8.7785	5.975
	3	8.7493	5.623
	4	8.7456	5.578
	5	8.8245	6.531
	6	8.7824	6.022
	7	8.7122	5.175
	8	8.7400	5.510
	9	8.7102	5.151

Table 2. Performance of each model on validation set. Since the actual optimality gaps of our models are less important than the comparison between them, we use Google OR-tools as a proxy of the optimal tour length.

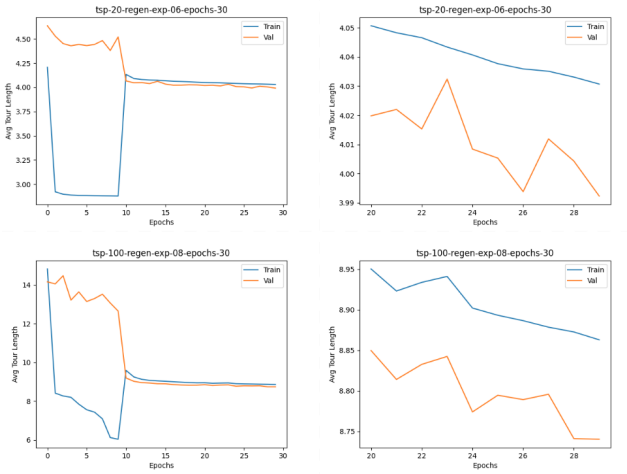

Figure 3. Training curves. At 10 epochs, the training distribution switches from the one specified by the curriculum to the uniform distribution, which is also the validation distribution. The left two curves show the training progress over all 30 epochs, while the right two curves show the training progress over the last 10 epochs. Note, on the right curves, the training and validation losses have not plateaued – this indicates that our models have not yet reached convergence. This effect is less pronounced when training on the uniform curriculum, which leads us to conclude that some of the discrepancies between uniform and curriculum tour lengths in tsp-100 can be explained by lack of convergence.

gogical distributions that help the training process but also if the method performs significantly better than random pedagogical distributions.

3.3. Automated Curriculum Learning with Fixed Pedagogical Distributions

We plan to evaluate the performance of our proposed method from Section 2.4 for automated curriculum learning in the setting where the set of potential pedagogical distributions is a fixed, discrete set. Here, we would use the human-made pedagogical distributions from Section 3.1. As a control, we would compare this method's performance to just training on the uniform distribution and to the hand-designed curriculums of Section 3.1. We also compare using the prediction gain progress signal with our novel signal test-aware prediction gain.

3.4. Fully Automated Curriculum Learning

We plan to evaluate the performance of our proposed method from Section 2.4 for automated curriculum learning in the setting where the set of pedagogical distributions is not fixed. In this setting, we proposed our action space to be all set of valid parameterizations P. We compare this method's performance to trainining on just the uniform distribution, hand-designed curriculums of Section 3.1, and automated curriculum learning with fixed subtasks of Section 3.3. We also continue the comparison between prediction gain and test-aware prediction gain.

3.5. Few-Shot Learning and Distribution Shift

Finally, we plan to evaluate the performance of the RL-based teacher methods (originally evaluated in Sections 3.3 and 3.4) in the settings of few-shot learning and distribution shift. To accomplish this, we consider two realistic scenarios involving the Chicago Taxi dataset (Chicago). This dataset includes the pickup and dropoff location/timing of taxi trips throughout Chicago (and can be used to represent data with patterns similar to that of the real-world). In one (corresponding to few-shot learning), we analyze how the RL-based teacher approaches can train without any real-world data other than a few samples from the Chicago Taxi dataset. We compare this with normally training on a large portion of the Chicago Taxi dataset and with training on each of the pedagogical distributions individually. With additional datasets, we could include real-world data from other cities in the pedagogical distributions. This is particularly realistic for the deployment of learned algorithms

in new settings. In the other scenario (corresponding to distribution shift), we analyze how the RL-based fixed pedagogical distribution teacher performs when the pedagogical distributions are data from different contiguous ranges of time. Distribution shifts are common (and those caused by COVID-19 are particularly harsh) and it is important that learning algorithms are able to utilize data from distributions as they shift to make the most informed decisions. It is our hope that in both of these scenarios, test-aware prediction gain will enable the RL-based teachers to identify pedagogical distributions that are most relevant to help supplement a lacking number of samples from the test distribution (due to few samples in total or distribution shift).

We note that some recent work such as (Delarue et al., 2020) takes a single-instance approach and does not train on distributions of instances. This is a particularly apt comparison bar for settings such as few-shot learning, but it is difficult as our results would be inherently tied to the underlying baseline architecture we use. Ultimately, our focus is how to best use these patters that arise through distributions.

4. Discussion and Future Work

In this work, we have proposed and evaluated methods for utilizing curriculum learning techniques to improve performance of learning algorithms on combinatorial optimization problems. Particularly, we focused on methods that are relevant to Euclidean TSP and VRP. Such methods are critical for real-world problems as they eliminate the need for algorithm design experts in some settings and help improve the efficiency of decisions that govern how we use limited resources (e.g. vehicles, energy). We examine our methods from the perspective of general improvement and the settings of few-shot learning and distribution shift. Current experimental results are promising, but are certainly inconclusive. Future work must be done to re-evaluate the experiments of Section 3.1, evaluate the experiments of Sections 3.2 to 3.5, and implement VRP experiments in addition to TSP experiments. We hope that future work can also utilize our methods to obtain performance improvements with different underlying baseline architectures. We emphasize that our techniques are completely agnostic to the underlying baseline architectures, and could be utilized even by architectures that handle billions of data points per instance such as (Drori et al.). It is this universality that provides encouragement our methods can help optimize many crucial real-world problems.

Finally, we encourage future work that analyzes the performance (and provides improvements) of our proposed methods and others on more adversarial instances. For example, (Papadimitriou & Steiglitz, 1978) provides examples where there are exponentially many local optima of poor quality that are very far away from better solutions. The emphasis on few-shot learning and distribution shift in our work is driven by a desire to highlight important dimensions for safe deployment of learned algorithms in the real-world. It is our hope that our methods and the field in general will make progress towards reliable learned solutions that deserve the trust of practitioners.

References

Applegate, D., Bixby, R., Chvatal, V., and Cook, W. Concorde tsp solver, 2006.

Bellemare, M., Srinivasan, S., Ostrovski, G., Schaul, T., Saxton, D., and Munos, R. Unifying count-based exploration and intrinsic motivation. *Advances in neural information processing systems*, 29:1471–1479, 2016.

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

Castells, T., Weinzaepfel, P., and Revaud, J. Superloss: A generic loss for robust curriculum learning. *Advances in Neural Information Processing Systems*, 33, 2020.

Chicago. *Chicago Taxi Trips Dataset*. URL `https://data.cityofchicago.org/Transportation/Taxi-Trips/wrvz-psew`.

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, 1976.

Costa, A. and Nannicini, G. Rbfopt: an open-source library for black-box optimization with costly function evaluations. *Mathematical Programming Computation*, 10(4): 597–629, 2018.

Delarue, A., Anderson, R., and Tjandraatmadja, C. Reinforcement learning with combinatorial actions: An application to vehicle routing, 2020.

Drori, I., Kates, B., Sickinger, W., Kharkar, A., Dietrich, B., Shporer, A., and Udell, M. Galaxytsp: A new billion-node benchmark for tsp. *World*, 3:1–904.

Drori, I., Kharkar, A., Sickinger, W. R., Kates, B., Ma, Q., Ge, S., Dolev, E., Dietrich, B., Williamson, D. P., and Udell, M. Learning to solve combinatorial optimization problems on real-world graphs in linear time. *arXiv preprint arXiv:2006.03750*, 2020.

Falkner, J. K. and Schmidt-Thieme, L. Learning to solve vehicle routing problems with time windows through joint attention. *arXiv preprint arXiv:2006.09100*, 2020.

Gao, L., Chen, M., Chen, Q., Luo, G., Zhu, N., and Liu, Z. Learn to design the heuristics for vehicle routing problem. *arXiv preprint arXiv:2002.08539*, 2020.

Graves, A., Bellemare, M. G., Menick, J., Munos, R., and Kavukcuoglu, K. Automated curriculum learning for neural networks. *arXiv preprint arXiv:1704.03003*, 2017.

Gurobi Optimization, L. Gurobi optimizer reference manual, 2020. URL `http://www.gurobi.com`.

Joshi, C. K., Cappart, Q., Rousseau, L.-M., Laurent, T., and Bresson, X. Learning tsp requires rethinking generalization. *arXiv preprint arXiv:2006.07054*, 2020.

Karlin, A. R., Klein, N., and Gharan, S. O. A (slightly) improved approximation algorithm for metric tsp. *arXiv preprint arXiv:2007.01409*, 2020.

Lee, J., Hwangbo, J., Wellhausen, L., Koltun, V., and Hutter, M. Learning quadrupedal locomotion over challenging terrain. *Science robotics*, 5(47), 2020.

Lenstra, J. K. and Kan, A. R. Complexity of vehicle routing and scheduling problems. *Networks*, 11(2):221–227, 1981.

Lisicki, M., Afkanpour, A., and Taylor, G. W. Evaluating curriculum learning strategies in neural combinatorial optimization. *arXiv preprint arXiv:2011.06188*, 2020.

Matiisen, T., Oliver, A., Cohen, T., and Schulman, J. Teacher-student curriculum learning. *IEEE transactions on neural networks and learning systems*, 2019.

McMenemy, P., Veerapen, N., Adair, J., and Ochoa, G. Rigorous performance analysis of state-of-the-art tsp heuristic solvers. In *European Conference on Evolutionary Computation in Combinatorial Optimization (Part of EvoStar)*, pp. 99–114. Springer, 2019.

Nazari, M., Oroojlooy, A., Snyder, L., and Takác, M. Reinforcement learning for solving the vehicle routing problem. In *Advances in Neural Information Processing Systems*, pp. 9839–9849, 2018.

Osaba, E., Martinez, A. D., Lobo, J. L., Laña, I., and Del Ser, J. On the transferability of knowledge among vehicle routing problems by using a cellular evolutionary multitasking. *arXiv preprint arXiv:2005.05066*, 2020.

Papadimitriou, C. H. and Steiglitz, K. Some examples of difficult traveling salesman problems. *Operations Research*, 26(3):434–443, 1978.

Peng, B., Wang, J., and Zhang, Z. A deep reinforcement learning algorithm using dynamic attention model for vehicle routing problems. In *International Symposium on Intelligence Computation and Applications*, pp. 636–650. Springer, 2019.

Perron, L. and Furnon, V. Or-tools. URL `https://developers.google.com/optimization/`.

Sun, Y., Ernst, A., Li, X., and Weiner, J. Generalization of machine learning for problem reduction: A case study on travelling salesman problems. *arXiv preprint arXiv:2005.05847*, 2020.

Vesselinova, N., Steinert, R., Perez-Ramirez, D. F., and Boman, M. Learning combinatorial optimization on graphs: A survey with applications to networking. *arXiv preprint arXiv:2005.11081*, 2020.

Zhang, X. and Yang, S. Learning (re-) starting solutions for vehicle routing problems. *arXiv preprint arXiv:2008.03424*, 2020.