Learning 2-opt Local Search from Heuristics as Expert Demonstrations

Citation for published version (APA):

Da Costa, P., Zhang, Y., Akcay, A., & Kaymak, U. (2021). Learning 2-opt Local Search from Heuristics as Expert Demonstrations. In *2021 International Joint Conference on Neural Networks (IJCNN)* [9533697] Institute of Electrical and Electronics Engineers. <https://doi.org/10.1109/IJCNN52387.2021.9533697>

DOI:

[10.1109/IJCNN52387.2021.9533697](https://doi.org/10.1109/IJCNN52387.2021.9533697)

Document status and date:

Published: 20/09/2021

Document Version:

Accepted manuscript including changes made at the peer-review stage

Please check the document version of this publication:

- A submitted manuscript is the version of the article upon submission and before peer-review. There can be important differences between the submitted version and the official published version of record. People interested in the research are advised to contact the author for the final version of the publication, or visit the DOI to the publisher's website.
- The final author version and the galley proof are versions of the publication after peer review.
- The final published version features the final layout of the paper including the volume, issue and page numbers.

[Link to publication](#)

General rights

Copyright and moral rights for the publications made accessible in the public portal are retained by the authors and/or other copyright owners and it is a condition of accessing publications that users recognise and abide by the legal requirements associated with these rights.

- Users may download and print one copy of any publication from the public portal for the purpose of private study or research.
- You may not further distribute the material or use it for any profit-making activity or commercial gain
- You may freely distribute the URL identifying the publication in the public portal.

If the publication is distributed under the terms of Article 25fa of the Dutch Copyright Act, indicated by the "Taverne" license above, please follow below link for the End User Agreement:

www.tue.nl/taverne

Take down policy

If you believe that this document breaches copyright please contact us at:

openaccess@tue.nl

providing details and we will investigate your claim.

Learning 2-opt Local Search from Heuristics as Expert Demonstrations

1st Paulo da Costa
School of Industrial Engineering
Eindhoven University of Technology
 Eindhoven, Netherlands
 p.r.d.oliveira.da.costa@tue.nl

2nd Yingqian Zhang
School of Industrial Engineering
Eindhoven University of Technology
 Eindhoven, Netherlands
 yqzhang@tue.nl

3rd Alp Akcay
School of Industrial Engineering
Eindhoven University of Technology
 Eindhoven, Netherlands
 a.e.akcay@tue.nl

4th Uzey Kaymak
School of Industrial Engineering
Eindhoven University of Technology
 Eindhoven, Netherlands
 u.kaymak@tue.nl

Abstract—Deep Reinforcement Learning (RL) has achieved high success in solving routing problems. However, state-of-the-art deep RL approaches require a considerable amount of data before they reach reasonable performance. This may be acceptable for small problems, but as instances grow bigger, this fact severely limits the applicability of these methods to many real-world instances. In this work, we study a setting where the agent can access data from previously handcrafted heuristics for the Traveling Salesman Problem. In our setting, the agent has access to demonstrations from 2-opt improvement policies. Our goal is to learn policies that can surpass the quality of the demonstrations while requiring fewer samples than pure RL. In this study, we propose to first learn policies with Imitation Learning (IL), leveraging a small set of demonstration data to accelerate policy learning. Afterward, we combine on policy and value approximation updates to improve performance over the expert’s performance. We show that our method learns good policies in a shorter time and using less data than classical policy gradient, which does not incorporate demonstration data into RL. Moreover, in terms of solution quality, it performs similarly to other state-of-the-art deep RL approaches.

I. INTRODUCTION

The Traveling Salesman Problem (TSP) is a well-known Combinatorial Optimization (CO) problem where the aim is to find an optimal tour that visits n locations once and returns to the origin. The TSP is known to be NP-hard, [1] and solving it optimally is usually achieved via integer linear programming and dynamic programming methods. However, solving large TSP instances optimally can be impractical due to high computational costs. For that reason, several (meta)heuristics have been proposed to solve the problem. Heuristics for the TSP can be classified in constructive and improvement methods. In the first, the goal is to compose a solution by iteratively extending a partial tour. In the latter, a complete solution is improved by certain operators that search for better solutions, for example, using k -opt edge swaps [2].

The work is partially supported by the NWO funded project Real-time data-driven maintenance logistics (project number: 628.009.012).

Fig. 1. A schematic representation of the proposed method. **a** A supervised learning (Behavior Cloning) policy is trained on data $\mathcal{D}_e$ made of trajectories from 2-opt local search heuristics acting as experts. **b** The trained policy in **a** is used to warm-start a Reinforcement Learning policy trained via Policy Gradient to maximize cost reductions of the next search steps.

Recently, using machine learning to solve CO problems has gained a lot of interest. For many problems, heuristics exist to make algorithmic decisions that otherwise would be too expensive to compute. This fact makes machine learning a viable option to make decisions in a more automated and optimized manner [3]. Thanks to the advances in deep learning, Reinforcement Learning (RL) methods have succeeded

in learning effective constructive and improvement policies for the TSP [4]–[10]. However, these methods require many steps of poor performance in simulation during training, partly due to their simple exploration rules, such as ϵ -greedy for value learning methods and noise-based exploration for policy learning methods.

Since most previous RL methods attempted to learn either improvement or constructive heuristics, it is natural to consider reusing expert information embedded in handcrafted heuristics to accelerate learning. In Imitation Learning (IL), the goal is precisely to reproduce the behavior of an expert policy in a sequential decision-making problem [11]. However, classical IL can only learn policies as good as expert policies [11]. In the case of CO problems, these expert policies are usually not optimal. When a suboptimal expert is available, policies learned with standard IL can be inferior to policies learned via RL with approaches such as policy gradient [12].

Moreover, in the case of expert heuristics policies, it is desirable to learn without online access to the policies given the high cost of computing expert heuristics rules. Thus, we focus on the case when a suboptimal improvement policy exists but can only be used to gather demonstrations, i.e., expert trajectories. Our objective is to incorporate the information from such demonstrations to learn faster than with pure RL and better than the expert policy for the TSP.

To achieve this goal, we propose to combine RL leveraging online interactions with a Euclidean TSP environment and a small number of demonstrations from 2-opt improvement heuristics. We combine a classical policy gradient objective with a previously trained policy via an offline supervised loss, leading the agent to prefer actions experienced in the demonstrations. Then, during its online phase, the method performs updates considering its self-generated data. Our method, depicted in Figure 1, is straightforward and can be applied to other CO problems where good heuristics already exist. Our method can outperform IL methods such as Behavior Cloning (BC) and pure policy gradient training in our experiments. Moreover, our method performs similar to a recently proposed RL method [10] but requires only 10% of the number of iterations to obtain comparable policies to other effective deep RL methods.

We summarize our main contributions as follows:

- We combine a classical policy gradient objective with a previously trained policy via behavior cloning on expert demonstrations.
- Our method yields better policies faster than with pure RL starting from suboptimal 2-opt heuristics as experts.
- We only make use of a small amount of demonstration data, retaining similar sample complexity to RL methods.

II. RELATED WORK

The TSP is a challenging problem in CO with applications in many domains [13]. Past successes in solving hard instances have been accredited to heuristics or a combination of heuristics and exact methods. Such heuristics include local search methods such as Lin-Kernighan-Helsgaun (LKH) [14]

and metaheuristics such as simulated annealing [15]. In common, these methods aim to reduce expensive computations by exploiting the structure of the problem combined with local and global neighborhood search.

Recently, deep learning has emerged as a viable option to solve routing problems [3]. Many works have considered approximating a function that either attempts to construct a solution or improve a given solution. These approaches resemble heuristics and, as such, can be classified in constructive and improvement methods. These methods have had considerable success, using either supervised or reinforcement learning.

a) Supervised Learning: Supervised Learning approaches for the TSP [4], [6] have considered the setting of given offline data containing optimal solutions as outputs. The goal is to learn a function that can reproduce optimal tours directly from node and distance data. Results show that supervised learning can be applied to this setup with fairly reasonable results. However, ensuring optimal labels for larger instances can be too computationally expensive. Note that these methods rely on a given optimal policy. Thus, they can also be seen as a type of Imitation Learning. However, we make a distinction to differentiate between learning from optimal and suboptimal strategies.

b) Reinforcement Learning: Model-free RL breaks the assumption of given optimal solutions [5], [7]–[10]. Previous methods used deep function approximation to learn directly from interactions. Most methods have focused on the setting where a solution has to be constructed sequentially, starting from a given location (node). In this case, training attempts to find policies that reduce the overall cost of a tour. In practice, to achieve good solutions, these methods have to sample multiple tours to find near-optimal ones [5], [7], [8].

Another approach considers learning over multiple steps, improving a given solution over a series of local operators [16]. For the TSP, these improvement-seeking methods have achieved good results when sampling a similar amount of operators as in construction methods [9], [10]. One major drawback of RL methods is their sample complexity, normally requiring many hours of training. Another aspect of previously proposed methods is that they learn from scratch, i.e., no previous policy is used to aid learning. As it is true that optimal solutions may not be available or expensive to compute, the same is not valid for heuristics. These are cheaper to compute than optimal solutions and can be used as a suboptimal expert to guide policy search. In this work, we focus our attention on recent improvement methods that learn over a class of 2-opt policies [9], [10]. In this case, heuristics already exist and can potentially be used to accelerate learning.

c) Imitation Learning: In Imitation Learning (IL), expert policies can be used as demonstrations of successful behavior. A simple approach to IL is known as Behavior Cloning (BC), which learns a policy through supervised learning on expert demonstrations [11]. Although BC has been used successfully in several instances it suffers from problems such as distribution shift between expert and learned policies [17]. Generative

Adversarial Imitation Learning (GAIL) [18] is a more recent approach that obtains performance gains over BC but requires an additional Generative Adversarial Network (GAN) [19] for training. Other approaches in IL considered online access to the expert to surpass the expert policy [12], [20], [21].

However, even when an expert is available for online interactions, as is the case of heuristics, querying expert heuristics can become computationally prohibitive. Thus, we consider a common case in CO, i.e., when expert demonstrations exist, but online interaction with the expert is not available or expensive. In CO, IL has been previously applied to learn branching strategies and accelerate branch and bound in the context of integer programming, [22]–[24] and predict the objective improvement of cutting planes selection in semidefinite programming problems [25].

d) Reinforcement Learning from Demonstrations: Methods combining RL with demonstrations have shown good results guiding policy learning in robotics and game playing. In [26], demonstration data was used to pretrain the policy network over expert player data. [27] proposed the Deep Q-learning from Demonstrations (DQfD) and stored the demonstrations in an experience replay buffer. [28], [29] proposed expanding policy gradient with additional GAIL regularization terms. Like GAIL, these require additional networks for generating expert-like policies. Instead, we focus on a simple BC initialization term that does not require additional neural networks and can be trained using a pre-existent policy network. Similar to our work, [30] use BC pretrained policies and Natural Policy Gradient (NPG) [31] with BC augmented loss to learn dexterous manipulation on robotics hands. Similarly, we propose to guide initial exploration via behavior cloning and combine it with on-policy policy gradients. We note that the general idea of bootstrapping RL with BC while learning TSP heuristics is not yet explored.

III. PRELIMINARIES

A. Traveling Salesman Problem

In the Euclidean TSP, given an input graph, represented as a sequence of n locations in a two-dimensional space $X = \{x_i\}_{i=1}^n$, where $x_i \in [0, 1]^2$, we are concerned with finding a permutation of the nodes, i.e. a solution $s = (\bar{s}_1, \dots, \bar{s}_n)$, that visits each node once (except the starting node) and has the minimum total length (cost). We define the cost of a solution as $L(s) = \|x_{\bar{s}_n} - x_{\bar{s}_1}\|_2 + \sum_{i=1}^{n-1} \|x_{\bar{s}_i} - x_{\bar{s}_{i+1}}\|_2$, where $\|\cdot\|_2$ denotes the ℓ_2 norm.

B. First and Best Improvement 2-opt Heuristics

General improvement heuristics enhance feasible solutions through a search procedure. Local search methods start at an initial solution and proceed to replace previous solutions with better solutions. In the effective Lin-Kernighan-Helsgaun (LKH) [14] heuristic for the TSP, the procedure searches for k edge swaps (k -opt moves) that will be replaced by new edges resulting in a shorter tour. A simpler version [32] considers 2-opt (Figure 2) and 3-opt moves alternatives as these balance solution quality and the $O(n^k)$ complexity of the moves.

Fig. 2. TSP solution before a 2-opt move (top), and after a best improvement move (3, 4) (bottom left) and a first improvement move (1, 3) (bottom right). Indices represent the order in which nodes are visited.

At each iteration of a 2-opt local search, two edges are selected to be deleted and replaced by two edges that result in a better tour. In doing so, 2-opt moves can be expressed by selecting two index positions $(\bar{a}_1, \bar{a}_2) = (i, j)$ of a tour s , i.e., $\bar{a}_1, \bar{a}_2 \in \{1, \dots, n\}, \bar{a}_1 < \bar{a}_2$, breaking edges between nodes at positions $(i-1, i)$ and $(j, j+1)$, inverting the tour between i and j and adding edges between $(i-1, j)$ and $(i, j+1)$. The selection of the edges is normally done by scanning the current solution for suitable edge pairs.

Since there can be many such pairs, a decision must be made over which move is considered first. Two well-studied choices are selecting the first set of edges at which an improvement is possible, i.e., First Improvement (FI) and the best reducing cost move, i.e., Best Improvement (BI). On average, selecting BI over FI gives worse results if the initial solution is chosen at random. However, when initialized with a greedy constructive heuristic, BI is better and faster on average [33].

C. Markov Decision Process

We adopt a standard Markov Decision Process (MDP) $\mathcal{M}$, defined by a tuple $\mathcal{M} = \langle \mathcal{S}, \mathcal{A}, \mathcal{P}, r, \gamma \rangle$ where $\mathcal{S}$ is the state space, $\mathcal{A}$ is the action space, $\mathcal{P}(s'|s, a)$ is the transition distribution after taking action a at state s , $r(s, a)$ is the reward function, $\gamma \in (0, 1)$ is a discount factor, detailed below.

1) *States:* A state s is a solution to the TSP defined by a permutation of n nodes. Note that this allows us to consider experts that only operate in s when picking the next action.

2) *Actions:* An action a is a tuple (i, j) , where $i, j \in \{1, \dots, n\}, i < j$ corresponding to two indices of s .

3) *Transitions:* Transitions are deterministic and defined by (s, a) pairs. Given action (i, j) , transitioning from $s = (s_1, \dots, s_i, \dots, s_j, \dots, s_n)$ is defined by a 2-opt exchange inverting the tour segment between i and j resulting in a next state $s' = (s_1, \dots, s_j, \dots, s_i, \dots, s_n)$.

4) *Objective*: Given a stochastic policy $\pi(a|s)^1 : \mathcal{S} \rightarrow \mathcal{P}(\mathcal{A})$ which maps states to action probabilities, its performance is evaluated by the expected discounted sum of rewards (return):

$$J(\pi) = \mathbb{E}_\pi[r(s, a)] = \mathbb{E}_{s_0, a_0, \dots} \left[\sum_{t=0}^{\infty} \gamma^t r(s_t, a_t) \right], \quad (1)$$

where $s_0, a_0, \dots$ is a trajectory induced by policy π , i.e. $s_0 \sim p_0(s_0)$, with $p_0(\cdot)$ being the distribution over initial states, $a_t \sim \pi(\cdot|s_t)$ and $s_{t+1} \sim \mathcal{P}(\cdot|s_t, a_t)$. We define standard characterizations of the value function $V^\pi(s) = \mathbb{E}_\pi[\sum_{k=0}^{\infty} \gamma^k r(s_{t+k}, a_{t+k}) | s_t = s]$, action value function $Q^\pi(s, a) = \mathbb{E}_\pi[\sum_{k=0}^{\infty} \gamma^k r(s_{t+k}, a_{t+k}) | s_t = s, a_t = a]$ and the advantage function as $A^\pi(s, a) = Q^\pi(s, a) - V^\pi(s)$, reflecting the expected additional return of a at state s .

5) *Rewards*: Rewards are attributed to actions that can improve upon the cost of the best-found state s_t^* in a trajectory, i.e. $s_t^* = \arg \min_{s_{t'} \in \{s_0, \dots, s_t\}} L(s_{t'})$ and $r(s_t, a_t) = L(s_t^*) - \min\{L(s_t^*), L(s_{t+1})\}$.

In RL, we are interested in finding a policy that maximizes $J(\pi)$ by using a set of trajectories $\mathcal{D} = \{\tau_i\}$, where $\tau_i = \{(s_0^i, a_0^i), (s_1^i, a_1^i), \dots\}$ are sampled from the current policy in on-policy methods or different policies in off-policy methods. We consider policies π_θ with parameters θ , thus, we abuse notation and use π , π_θ and $J(\pi)$, $J(\theta)$ interchangeably.

IV. LEARNING 2-OPT FROM DEMONSTRATIONS

In this work, we use a combination of RL and IL to learn 2-opt exchange heuristics. To reduce sample complexity and help exploration, we collect expert demonstrations from FI and BI heuristics, extract a policy from demonstrations and do policy gradient updates over environment interactions. In the next sections, we detail each component of the proposed method.

A. Policy Gradient

We are mainly concerned with policy gradient methods, which are a class of on-policy model-free RL. In policy gradient, the parameters of the policy are directly optimized towards maximizing the main objective defined as $J(\theta) = \mathbb{E}_{s \sim d^\pi} [\sum_{a \in \mathcal{A}} \pi_\theta(a|s) Q^\pi(s, a)]$, where $d^\pi(s)$ is the stationary distribution of the Markov chain induced by $\pi_\theta(a|s)$. Then, according to the policy gradient theorem [34], the gradient of $J(\theta)$ can be estimated as:

$$\nabla_\theta J(\theta) \propto \sum_{s \in \mathcal{S}} d^\pi(s) \sum_{a \in \mathcal{A}} \nabla_\theta \pi_\theta(a|s) Q^\pi(s, a) \quad (2)$$

where θ can be optimized via gradient ascent. In practice, to reduce variance, $Q^\pi(s, a)$ is replaced by the advantage function $A^\pi(s, a)$ [35]. In this work, we consider the REINFORCE algorithm [36], thus, during training we collect on-policy trajectories $\mathcal{D}^\pi$ and replace expectations by empirical samples to compute gradient estimates as:

$$g \propto \sum_{\tau_i \in \mathcal{D}^\pi} \sum_t \nabla_\theta \log \pi_\theta(a_t^i | s_t^i) \hat{A}^\pi(s_t^i, a_t^i) \quad (3)$$

¹We refer to deterministic policies as $\pi(s)$.

where $\hat{A}^\pi(s_t, a_t) = G_t - V_\phi^\pi(s_t)$ is an estimate of the advantage function, $G_t = r(s_t, a_t) + \sum_{k=1}^T \gamma^k r(s_{t+k}, a_{t+k})$ are Monte Carlo estimates of returns up until T and $V_\phi^\pi(s_t)$ is an approximation of $V^\pi(s_t)$ with parameters ϕ trained over mean squared errors between G_t and $V_\phi^\pi(s)$.

B. Learning from Demonstrations

Directly optimizing policy gradients with rewards defined in $\mathcal{M}$ can lead to good policies that surpass simple greedy 2-opt heuristics [9], [10]. However, doing so requires a large number of samples and many hours of training. Demonstrations can help alleviate this issue and help to guide exploration to good reward regions. In this work, we consider demonstrations from deterministic 2-opt policies FI and BI that select actions from states as $a = \pi^e(s)$, where π^e is the expert's policy.

Note that the objective in $\mathcal{M}$ recovers the total improvement over an initial state, albeit the discount factor. Thus, having expert demonstrations from greedy heuristics can help to find regions with higher improvement. Moreover, the BI heuristic is the optimal policy of a one-step $\mathcal{M}$. Therefore, quickly extracting information from this policy can potentially help to guide policies over larger horizons.

Algorithm 1: Policy gradient with behavior cloning

Input: Expert demonstrations $\mathcal{D}^e$; parameters θ, ϕ , weight β ; maximum steps $\mathbb{T} \geq T$, number of epochs m and iterations L, K .

```

for  $l = 1, 2, \dots, L$  do
  Update  $\theta$  with demonstrations for  $m$  epochs by:
     $\sum_{(s,a) \in \mathcal{D}^e} \nabla_\theta \log \pi_\theta(a|s)$ ;
end
for  $k = 1, 2, \dots, K$  do
  Sample  $\mathcal{D}^\pi = \{\tau_i\}_{i=1}^N, \tau_i \sim \pi_\theta$ ;
  Update  $\theta$  and  $\phi$  every  $T$  steps by:
     $\sum_{(s,a) \in \mathcal{D}^\pi} \nabla_\theta \log \pi_\theta(a|s) \hat{A}^\pi(s, a) + \beta \nabla_\theta H(\theta)$ ;
     $\frac{1}{2} \nabla_\phi \sum_{(s,a) \in \mathcal{D}^\pi} \|G_t - V_\phi^\pi(s)\|_2^2$ ;
end

```

1) *Behavior Cloning*: Exploration in policy gradient methods is done by implicitly using the stochasticity of policies or by explicitly introducing an entropy term to the objective. If the initial policy is poor, learning can be slow, as the algorithm explores states that lead to poor rewards. An effective way to combat this issue is to use expert policies and attempt to mimic their behavior. Behavior Cloning (BC) is a simple IL method that attempts to learn good policies over expert trajectories $\mathcal{D}^e$ and does not require additional interactions with the expert. The main objective in BC is to train a policy π_θ on a supervised signal from a distribution of states $d^{\pi^e}(s)$ induced by the expert policy π^e defined as $C(\theta) = \mathbb{E}_{s \sim d^{\pi^e}} [\ell(\pi_\theta(\pi^e(s)|s), \pi^e(s))]$, where $\ell(\cdot)$ is a suitable performance loss. In our case, we approximate the objective above over state-action pairs in $\mathcal{D}^e$ by the log-likelihood loss as

$$C(\theta) \propto \sum_{\tau_i \in \mathcal{D}^e} \sum_t \log \pi_\theta(a_t^i | s_t^i). \quad (4)$$

Taking the gradient of this objective resembles the one in (3), but here we average over expert and not on-policy trajectories and do not have access to an advantage function. In fact, (4) can perform well when states observed by expert policies are similar to states observed by learned policies [11]. The BC policy is then the one that finds parameters θ such that $C(\theta)$ is maximized and the distribution of actions given a state approaches those on demonstration data. In our study, each trajectory demonstration τ_i^e relates to a TSP instance, and cloning a policy corresponds to cloning different actions in different instances. Since states visited by the expert policy are diverse, we expect BC to perform well in the given setting, assuming we can replicate the expert’s policy. Moreover, even if learning fails, the BC policy may still perform well by encountering alternative good decisions with a low probability under expert trajectories. Still, high rewards under the performance of (1) [3], which is ultimately what we care about. However, in general, BC cannot outperform the expert. Thus, we still need exploration to perform better than the expert.

2) *Guiding Exploration with Behavior Cloning*: As demonstrated in [30], we can use demonstration data to both provide good initialization for RL or use them to guide exploration during RL. In our case, as the number of iterations grow the policy is given a chance to let go of demonstrations and perform policy updates only on on-policy samples aided by an entropy term $H(\theta) \propto \beta \sum_{s \in \mathcal{D}^\pi} \mathbb{E}_{a \sim \pi_\theta} [\log \pi_\theta(\cdot|s)]$, where $\beta \in [0, 1)$. A pseudocode of the procedure is presented in Algorithm 1.

Fig. 3. A state (solution) s is fed to an encoder where graph and sequence information are extracted. A policy decoder takes encoded inputs to output actions sequentially. A value decoder operates outputs state values estimates.

C. Policy and Value Networks

We adopt the effective neural network architecture reported in [10]. In the paper, an encoder-decoder architecture is proposed to output estimates of both $\pi_\theta(a|s)$ and $V_\phi^\pi(s)$ for a given state s^2 . However, in the original paper, a state is composed of the tuple (s_t, s_t^*) whereas we only consider a solution s_t . Although this modification makes convergence slower, this allows us to have comparable state distributions between expert and online policies since $s_t = s_t^*$ for both FI and BI.

In the architecture, depicted in Figure 3, the shared encoder is composed of three Graph Convolutional [37] layers to extract node information and a Bi-directional Long Short-Term Network (LSTM) [38] layer responsible for encoding tour sequence information. The policy decoder then uses both graph

and sequence information to generate actions sequentially via a pointing attention mechanism [4]. That is, it generates 2-opt indices sampling over softmax operators $p_\theta(\cdot)$ and uses the chain rule to factorize $\pi_\theta(a|s)$ as $\pi_\theta(a|s) = p_\theta(\bar{a}_1|s)p_\theta(\bar{a}_2|\bar{a}_1, s)$, where, $a = (\bar{a}_1, \bar{a}_2)$, $\bar{a}_i \in \{1, \dots, n\}$. The value decoder employs two dense layers that take combined graph and sequence representations to output value estimations. We refer the reader to [10] for details on the original implementation.

V. EXPERIMENTS

This section aims to investigate whether initializing policy gradient with a BC policy can help with learning 2-opt heuristics faster than a method that uses no previous demonstration data. We conduct extensive experiments to investigate the performance of warm starting policy gradient with BC (BC+PG) to a Policy Gradient (PG) policy. We consider three tasks in our experiments, Euclidean TSP instances with 20 (TSP20), 50 (TSP50), and 100 nodes (TSP100). For all tasks, node coordinates are drawn uniformly at random in the unit square $[0, 1]^2$. We measure the performance of policies on the optimality gap between the best-found solution and the optimal solution computed using Concorde [39]. We compare our results on the test dataset as reported in [8] containing 10,000 instances for each task. To benchmark our method, we compare learned policies to other highly specialized RL methods for the TSP. Here the objective is not only achieving near-optimal solutions but comparing how achieving a certain performance level depends on the number of samples.

A. Experimental Settings

In our experiments, a maximum of 100,000 samples of demonstration data is gathered using either FI or BI on 5,120 TSP instances over trajectories of a maximum of 400 steps. When this procedure generated more than 100,000 samples, we undersampled the expert samples uniformly at random. Each experiment block is repeated for FI and BI demonstrations separately. During RL, 5,120 instances of the TSP were generated on the fly and simulated for $T = 200$ steps. Every T steps policy and value updates were performed, where returns are computed over a maximum of $T = 8$ steps in the future (truncation). In all tasks, we pass twice over the expert demonstrations i.e. $m = 2$ for each iteration $l \in \{1, \dots, 200\}$ and use a batch size of 512 during BC, and $k \in \{1, \dots, 200\}$ iterations over a batch size of $512 \times T$ samples during RL. During BC, a teacher forcing [40] ratio of 25% is used to accelerate learning. A fixed validation dataset of 128 instances with their respective optimal solutions was used for manual hyperparameter optimization rolling out policies for 200 steps.

Following the implementation in [10], we employed the same $\gamma = 0.99$, ℓ_2 penalty of 1×10^{-5} and learning rate $\lambda = 0.001$, λ decaying by 0.98 at each k . Loss weights $\beta = 0.0045$ decay by 0.9 after every iteration, and parameter updates were performed via Adam [41]. The remaining neural network hyperparameters were unchanged from the original implementation. We train on an RTX 2080Ti GPU and Ryzen 3950X CPU hardware using PyTorch 1.6. Each iteration taking

²A portion of parameters θ and ϕ are shared in the encoding layers.

Fig. 4. Optimality gaps when rolling out policies for 200 steps on 128 instances using Best Improvement (BI) demonstrations. **Policies:** Expert, Behavior Cloning (BC), Behavior Cloning followed by Policy Gradient (BC+PG), and Policy Gradient (PG).

Fig. 5. Optimality gaps when rolling out policies for 200 steps on 128 instances using First Improvement (FI) demonstrations. **Policies:** Expert, Behavior Cloning (BC), Behavior Cloning followed by Policy Gradient (BC+PG), and Policy Gradient (PG).

an average time of 1m 52s, 2m 23s, and 3m 58s for TSP20, TSP50, and TSP100 during the RL phase. We roll out our policy for 1,000 steps during testing to allow for a fair comparison with previous literature. Our implementation will be made available online.

B. Experimental Results and Discussion

We present the optimality gaps of the tested policies using BI and FI demonstrations in the log scale in Figures 4 and 5. In Figure 4, we observe that for all tested instances, warm starting PG with a cloned policy using BC yields better solutions (lower optimality gaps). The difference is more prominent at early iterations due to PG having to sample more exploratory actions. Aided by BC, BC+PG can, in fewer iterations, surpass the performance of the BC policy and the expert. For example, in TSP20 instances, BC+PG can surpass the expert in one iteration, whereas PG requires 10 iterations. To reach a 2% optimality gap in TSP50, PG requires 80 iterations, whereas BC+PG requires 40 iterations, i.e., around 0.42×10^8 samples. Results are similar for TSP 100, where we observe that it takes only 5 iterations (0.06×10^8 samples) for BC+PG to reach a 20% gap, whereas PG requires 20 iterations.

In our MDP, increasing the size of instances also corresponds to increasing the size of the action space by order $O(n^2)$, thus, requiring further exploration of good actions before improvements can be made. We note that the performance of BC decreases with the instance size. BC can surpass the expert policy for TSP20 instances due to sampling more

actions during validation. For larger instances, BC policies cannot reach the same performance as the expert, although they still aid PG to achieve better performance. Note that BC also needs to be trained via supervised learning but requires few iterations to converge, and it is much faster to run than PG. On average, a BC policy trained over BI trajectories converged after 0.13×10^8 samples from the 100,000 expert state-action pairs. In terms of running time, learning via BC on TSP100 instances took 28s per iteration, i.e., 12% of the time of the PG counterpart. A similar reduction in time is seen for TSP20 and TSP50. BC convergence is achieved between 50 and 100 iterations, thus requiring 7% of the time to run PG.

In Figure 5, we note that as the size of instances increases, the performance of BC decreases considerably. This result is expected as actions become harder to predict given the size of the instance and show that learning directly from FI demonstrations is harder than from BI. We argue that this difference comes from the more diverse and clear action selection of BI, where actions are selected based on the best improvement move, yielding diverse but consistent action indices each time. FI, on the other hand, selects many moves that swap the first node in the tour, resulting in many actions of the form $a = (1, j)$, where $j > 1$, i.e., at each iteration it starts scanning a solution from the beginning of a tour returning the first improvement encountered. While this is clear from a heuristic perspective, our architecture cannot appropriately learn this action selection rule for large instances. Nevertheless, learning

TABLE I
PERFORMANCE W.R.T. CONCORDE. *S*: NUMBER OF SAMPLES. *Type*: SOLVER, **SL**: SUPERVISED LEARNING, **BC**: BEHAVIOR CLONING, **RL**: REINFORCEMENT LEARNING.

Method	Type	TSP20			TSP50			TSP100		
		Cost	Gap	$S (\times 10^8)$	Cost	Gap	$S (\times 10^8)$	Cost	Gap	$S (\times 10^8)$
Concorde	Solver	3.84	0.00%	-	5.70	0.00%	-	7.76	0.00%	-
GCN	SL	3.84	0.01%	0.15	5.70	0.01%	0.15	7.87	1.39%	0.15
GAT-C	RL	3.84	0.09%	0.05	5.75	1.00%	0.05	8.12	4.64%	0.05
AM-C	RL	3.84	0.08%	1.28	5.73	0.52%	1.28	7.94	2.26%	1.28
GAT-I	RL	3.84	0.03%	4.09	5.75	0.83%	4.09	8.01	3.24%	4.09
GCN-I	RL	3.84	0.00%	2.05	5.71	0.21%	3.07	7.86	1.26%	3.07
PG@1	RL	7.62	98.72%	0.01	14.08	147.30%	0.01	33.66	333.50%	0.01
PG@5	RL	4.04	5.26%	0.05	7.59	33.30%	0.05	11.06	42.50%	0.05
PG@20	RL	3.85	0.21%	0.21	5.94	4.28%	0.21	8.56	10.29%	0.21
PG@200	RL	3.84	0.01%	2.05	5.71	0.30%	2.05	7.89	1.61%	2.05
BC+PG@1	BC, RL	3.88	1.17%	0.01	6.28	10.29%	0.01	8.86	14.15%	0.01
BC+PG@5	BC, RL	3.84	0.07%	0.05	5.84	2.61%	0.05	8.47	9.02%	0.05
BC+PG@20	BC, RL	3.84	0.02%	0.21	5.74	0.86%	0.21	8.07	3.98%	0.21
BC+PG@200	BC, RL	3.84	0.00%	2.05	5.71	0.21%	2.05	7.87	1.41%	2.05

a BC policy from FI demonstrations can still help to converge to better policies than with vanilla PG.

C. Comparison to Exact and Previous Learning Methods

We report results on the same 10,000 instances for each TSP size as in [8] and optimal costs obtained by Concorde [39]. We include recent state-of-the-art deep learning methods, including supervised learning [6], construction [7], [8] and improvement [9], [10] reinforcement learning methods. We present the best-reported results using sampling for constructive and supervised learning methods, as these yield the lowest costs. We report results after rollouts of 1000 steps for improvement methods, as these are similar to sampling in constructive methods. We note that the test data used in [6], [8], [10] is the same and, therefore, results are directly comparable. In [9], the generation process and size are identical, which decreases the variance of the results.

Since we are mainly interested in obtaining good policies quicker than using only RL, we show the performance of trained policies at different iterations. We refer to our tested policies as PG@ k and BC+PG@ k , where k is the iteration number to return the best-observed policy on validation instances. We report BC+PG results from BI demonstrations as these resulted in better performance. We refer to the results [6] as GCN in [8] as AM-C, in [7] as GAT-C, in [9] as GAT-I, and in [10] as GCN-I. We point out that our PG uses a simplified version of GCN-I that runs for fewer iterations for TSP50 and TSP100 and leverages just the current solution for action selection. We select GCN-I as a baseline for complexity as it is more sample efficient than [9]. Comparison results, including cost, optimality gaps and sample complexity³ are summarized in Table I.

a) Solution Quality: We observe in the results that BC+PG generates better policies than PG over the entire training procedure, i.e., at $k = 200$. For example, in TSP50,

BC+PG@200 results in a 0.21% optimality gap, whereas PG@200 results in 0.30%. Moreover, the difference between the two policies is larger in the early stages of training. This is expected as after BC is over, updates to the policy follow the same gradient updates. When we compare PG with GCN-I, we see the effect of simplifying the neural network, reducing the number of samples, and fixating the planning horizon. The simplification results in larger gaps than the original model. BC+PG brings the optimality gaps much closer to GCN-I, matching it for TSP20 and TSP50. In other words, starting from a cloned policy helps to achieve similar performance as a more expressive model trained over more samples.

b) Early Policies: When comparing early policies, i.e., $k \in \{1, 5, 20\}$, we note the benefit of biasing with BC pretraining at the early stages of RL training. Those policies can achieve much lower gaps than PG at the same k , e.g., BC+PG@1 achieves a 1.17% gap, whereas the same iteration in PG has a 98.72% gap. Compared to previous methods, we expect supervised learning and construction methods to have lower sample complexity than improvement methods, e.g., GCN, GAT-C, and AM-C. Interestingly, early policies learned at PG+BC@20 are competitive with previous RL methods that sample a much larger number of samples. Even if we add the number of samples to convergence during BC ($\sim 0.13 \times 10^8$) to 0.21×10^8 , we observe that these policies are not far from results obtained with AM-C and GAT-I sampling 1.28 and 4.09×10^8 data points, respectively. For example, in TSP50, BC+PG@20 achieves a 0.86% gap requiring, whereas GAT-I achieves a 0.83% in the same task.

VI. CONCLUSION AND FUTURE WORK

In this work, we studied learning 2-opt local search heuristics for the Traveling Salesman Problem (TSP) given greedy heuristics as expert demonstration data. We use the demonstration data to provide good initialization to a reinforcement learning algorithm based on policy gradient. We propose to

³The RL portion of BC+PG.

initialize policy gradient from imitating expert policies using behavior cloning. Our results show that demonstrations are beneficial at the beginning of RL training leading to good policies quicker than with vanilla policy gradient. Moreover, when training is initialized with behavior cloning, final policies at the end of learning generate TSP tours with lower optimality gaps. We show that policies learned with few policy gradient iterations are competitive with previous deep learning models proposed and can be obtained with a much lower sample complexity than previous models. Future research could examine directions to improve imitation and reinforcement learning performance in heuristics in combinatorial optimization settings. This is a relevant research direction and important to more general combinatorial problems.

REFERENCES

- [1] C. H. Papadimitriou, "The euclidean travelling salesman problem is np-complete," *Theoretical Computer Science*, vol. 4, no. 3, pp. 237–244, 1977.
- [2] K. Helsgaun, "An effective implementation of the lin-kernighan traveling salesman heuristic," *European Journal of Operational Research*, vol. 126, no. 1, pp. 106–130, 2000.
- [3] Y. Bengio, A. Lodi, and A. Prouvost, "Machine learning for combinatorial optimization: a methodological tour d'horizon," *arXiv preprint arXiv:1811.06128*, 2018.
- [4] O. Vinyals, M. Fortunato, and N. Jaitly, "Pointer networks," in *Proceedings of the 29th Conference on Neural Information Processing Systems (NIPS)*, 2015, pp. 2692–2700.
- [5] I. Bello and H. Pham, "Neural combinatorial optimization with reinforcement learning," in *Proceedings of the 5th International Conference on Learning Representations (ICLR)*, 2017.
- [6] C. K. Joshi, T. Laurent, and X. Bresson, "An efficient graph convolutional network technique for the travelling salesman problem," *arXiv:1906.01227*, 2019.
- [7] M. Deudon, P. Cournut, A. Lacoste, Y. Adulyasak, and L.-M. Rousseau, "Learning heuristics for the tsp by policy gradient," in *Proceedings of the 15th International Conference on the Integration of Constraint Programming, Artificial Intelligence, and Operations Research (CPAIOR)*, 2018, pp. 170–181.
- [8] W. Kool, H. van Hoof, and M. Welling, "Attention, learn to solve routing problems!" in *Proceedings of the 7th International Conference on Learning Representations (ICLR)*, 2019.
- [9] Y. Wu, W. Song, Z. Cao, J. Zhang, and A. Lim, "Learning improvement heuristics for solving the travelling salesman problem," *arXiv:1912.05784*, 2019.
- [10] P. R. de O. da Costa, J. Rhuggenaath, Y. Zhang, and A. Akcay, "Learning 2-opt heuristics for the traveling salesman problem via deep reinforcement learning," 2020.
- [11] S. Ross and D. Bagnell, "Efficient reductions for imitation learning," in *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, 2010, pp. 661–668.
- [12] C.-A. Cheng, X. Yan, N. Wagener, and B. Boots, "Fast policy learning through imitation and reinforcement," *arXiv preprint arXiv:1805.10413*, 2018.
- [13] J. K. Lenstra and A. R. Kan, "Some simple applications of the traveling salesman problem," *Journal of the Operational Research Society*, vol. 26, no. 4, pp. 717–733, 1975.
- [14] K. Helsgaun, "General k-opt submoves for the lin-kernighan tsp heuristic," *Mathematical Programming Computation*, vol. 1, no. 2-3, pp. 119–163, 2009.
- [15] P. J. Van Laarhoven and E. H. Aarts, "Simulated annealing," in *Simulated annealing: Theory and applications*. Springer, 1987, pp. 7–15.
- [16] X. Chen and Y. Tian, "Learning to perform local rewriting for combinatorial optimization," in *Advances in Neural Information Processing Systems*, 2019, pp. 6281–6292.
- [17] S. Ross, G. Gordon, and D. Bagnell, "A reduction of imitation learning and structured prediction to no-regret online learning," in *Proceedings of the fourteenth international conference on artificial intelligence and statistics*, 2011, pp. 627–635.
- [18] J. Ho and S. Ermon, "Generative adversarial imitation learning," in *Advances in neural information processing systems*, 2016, pp. 4565–4573.
- [19] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial nets," in *Advances in neural information processing systems*, 2014, pp. 2672–2680.
- [20] K.-W. Chang, A. Krishnamurthy, A. Agarwal, H. Daume, and J. Langford, "Learning to search better than your teacher," in *International Conference on Machine Learning*. PMLR, 2015, pp. 2058–2066.
- [21] W. Sun, J. A. Bagnell, and B. Boots, "Truncated horizon policy search: Combining reinforcement learning & imitation learning," *arXiv preprint arXiv:1805.11240*, 2018.
- [22] A. Marcos Alvarez, L. Wehenkel, and Q. Louveaux, "Online learning for strong branching approximation in branch-and-bound," 2016.
- [23] E. B. Khalil, P. Le Bodic, L. Song, G. Nemhauser, and B. Dilkina, "Learning to branch in mixed integer programming," in *Thirtieth AAAI Conference on Artificial Intelligence*, 2016.
- [24] M. Gasse, D. Chételat, N. Ferroni, L. Charlin, and A. Lodi, "Exact combinatorial optimization with graph convolutional neural networks," in *Advances in Neural Information Processing Systems*, 2019, pp. 15 580–15 592.
- [25] R. Baltean-Lugoian, P. Bonami, R. Misener, and A. Tramontani, "Scoring positive semidefinite cutting planes for quadratic optimization via trained neural networks," 2019.
- [26] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot *et al.*, "Mastering the game of go with deep neural networks and tree search," *nature*, vol. 529, no. 7587, pp. 484–489, 2016.
- [27] T. Hester, M. Vecerik, O. Pietquin, M. Lanctot, T. Schaul, B. Piot, D. Horgan, J. Quan, A. Sendonaris, G. Dulac-Arnold *et al.*, "Deep q-learning from demonstrations," *arXiv preprint arXiv:1704.03732*, 2017.
- [28] J. Chen and W. Xu, "Policy gradient from demonstration and curiosity," 2020.
- [29] B. Kang, Z. Jie, and J. Feng, "Policy optimization with demonstrations," ser. Proceedings of Machine Learning Research, J. Dy and A. Krause, Eds., vol. 80. Stockholmsmässan, Stockholm Sweden: PMLR, 10–15 Jul 2018, pp. 2469–2478. [Online]. Available: <http://proceedings.mlr.press/v80/kang18a.html>
- [30] A. Rajeswaran, V. Kumar, A. Gupta, G. Vezzani, J. Schulman, E. Todorov, and S. Levine, "Learning complex dexterous manipulation with deep reinforcement learning and demonstrations," *Robotics: Science and Systems XIV*, Jun 2018. [Online]. Available: <http://dx.doi.org/10.15607/RSS.2018.XIV.049>
- [31] S. M. Kakade, "A natural policy gradient," in *Advances in neural information processing systems*, 2002, pp. 1531–1538.
- [32] S. Lin and B. W. Kernighan, "An effective heuristic algorithm for the traveling-salesman problem," *Operations research*, vol. 21, no. 2, pp. 498–516, 1973.
- [33] P. Hansen and N. Mladenović, "First vs. best improvement: An empirical study," *Discrete Applied Mathematics*, vol. 154, no. 5, pp. 802–817, 2006.
- [34] R. S. Sutton, D. A. McAllester, S. P. Singh, and Y. Mansour, "Policy gradient methods for reinforcement learning with function approximation," in *Advances in neural information processing systems*, 2000, pp. 1057–1063.
- [35] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. MIT press, 2018.
- [36] R. J. Williams, "Simple statistical gradient-following algorithms for connectionist reinforcement learning," *Machine learning*, vol. 8, no. 3-4, pp. 229–256, 1992.
- [37] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," in *Proceedings of the 5th International Conference on Learning Representations (ICLR)*, 2017.
- [38] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.
- [39] D. Applegate, R. Bixby, V. Chvatal, and W. Cook, "Concorde tsp solver," URL <http://www.math.uwaterloo.ca/tsp/concorde>, 2006.
- [40] R. J. Williams and D. Zipser, "A learning algorithm for continually running fully recurrent neural networks," *Neural computation*, vol. 1, no. 2, pp. 270–280, 1989.
- [41] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," in *International Conference on Machine Learning*, 2015.