Deep-Steiner: Learning to Solve the Euclidean Steiner Tree Problem

Siqi Wang(✉), Yifan Wang, and Guangmo Tong

University of Delaware, Newark, DE 19716, USA
{wsqbit,yifanw,amotong}@udel.edu

Abstract. The Euclidean Steiner tree problem seeks the min-cost network to connect a collection of target locations, and it underlies many applications of wireless networks. In this paper, we present a study on solving the Euclidean Steiner tree problem using reinforcement learning enhanced by graph representation learning. Different from the commonly studied connectivity problems like travelling salesman problem or vehicle routing problem where the search space is finite, the Euclidean Steiner tree problem requires to search over the entire Euclidean space, thereby making the existing methods not applicable. In this paper, we design discretization methods by leveraging the unique characteristics of the Steiner tree, and propose new training schemes for handling the dynamic Steiner points emerging during the incremental construction. Our design is examined through a sanity check using experiments on a collection of datasets, with encouraging results demonstrating the utility of our method as an alternative to classic combinatorial methods.

Keywords: Reinforcement learning · Combinatorial optimization · The steiner tree problem

1 Introduction

Network connection is an important issue in wireless networks [8,15,35], and a key problem in such studies is to build a connected network with the minimum cost, for example, the location-selection problem [31], the relay node placement problem [10], and the network connectivity restoring problem [39]. Such problems can often be nicely reduced to the classic Euclidean Steiner tree (EST) problem, where we seek to connect a collection of points in the Euclidean space using Steiner minimal trees. The EST problem is well-known to be NP-hard, even in the two-dimensional Euclidean space [16], and various algorithms have been developed for effective and efficient solutions [3,13,22,25,40,41,47]. Recently, a trending direction is to design reinforcement learning diagrams for solving combinatorial optimization problems in a data-driven manner. This is driven by as least two compelling reasons: philosophically, we are wondering if effective heuristics can be automatically inferred by artificial intelligence [34]; practically, heuristics produced by reinforcement learning are tuned by the empirical distribution of the inputs, and can therefore offer better performance for a specific

Z. J. Haas et al. (Eds.): WiCON 2022, LNICST 464, pp. 228–242, 2023.
https://doi.org/10.1007/978-3-031-27041-3_16

application scenario [5] [34]. In this paper, we revisit the EST problem and design reinforcement learning methods.

Challenges. Learning-based methods have recently shown promising results in solving combinatorial optimization problems [5,26,28]. In principle, searching the best solution to combinatorial optimization problems can be modeled as a Markov decision process (MDP) in which the optimal policy is acquired using reinforcement learning [11]. Enhanced by deep learning techniques, the policies can be further parameterized through neural networks for learning better representation [28] [36]. The existing works have studied problems like the travelling salesman problem (TSP) and the vehicle routing problem where the searching space is finite, and therefore, they focus primarily on policy parameterization and training [28] [36]. The key challenge in solving the EST problem is that the searching space is unstructured: the Steiner points could be any subset of the Euclidean space. Therefore, as opposed to the TSP problem where the searching space is composed of all permutations of the nodes, the solutions to the EST problem cannot be readily enumerated. A straightforward method is to discretize the Euclidean space using grid networks by which the Steiner points are assumed to be on the joints. However, such a simple method is intuitively not optimal because it is unable to take account of any combinatorial properties of the Steiner trees. In this paper, we make an attempt to address the above challenges by presenting a reinforcement learning framework called Deep-Steiner for solving the EST problem. Our work in this paper can be summarized as follows:

- **Space discretization.** Leveraging the unique properties of Steiner trees, we design methods for creating a compact candidate set for the Steiner points. The resulted candidate set is dynamically constructed during the incremental searching, thereby being superior to methods adopting fixed searching spaces (e.g., grid network). In addition, the complexity of our candidate set can be easily tuned through hyper-parameters, admitting a controllable efficacy-efficiency trade-off.
- **Policy parameterization and training schemes.** We present methods for parameterizing the searching policy using attention techniques to dynamically update the point and graph embeddings. Our methods iteratively update the point and graph embeddings and then select a Steiner point until meeting the stopping criterion, which is different from previous methods that only compute the embeddings once. In addition, we design new reinforcement schemes for parameter training, where three stopping criteria are proposed for different training purposes.
- **Implementation and experiments.** Extensive experiments have been conducted to examine the proposed methods on different datasets and different training schemes. We have acquired encouraging results showing that the proposed method is non-trivially better than the baselines, which might be the first piece of evidence that demonstrates the potential of data-driven approaches for addressing combinatorial optimization problems with complex searching spaces. Our source code is made publicly available[1].

[1] https://github.com/EricW1996/EAI-Deepsteiner.

2 Related Work

Steiner Tree in Wireless Networks. In the field of wireless network, many research problems are closely related to the Steiner tree problem. For example, the relay node placement problem can be reduced to the EST problem with bounded edge length [10,30]. Furthermore, solving the EST problem is also a key to many other problems, including the problem of restoring the network connectivity [39], the minimum connected dominating set problem [33], the problem of broadcast routing [29], and the minimum length multicast tree problem [18].

Traditional Methods. Traditional approaches for solving the EST problem can be classified into three branches: exact algorithms, approximation algorithms, and heuristic methods. Warme *et al.* propose the GeoSteiner algorithm [45], which is the most successful exact algorithm for the EST problem. Their algorithm is based on the generation and concentration of the full Steiner tree. For heuristic algorithms, Thompson *et al.* provide an edge insertion algorithm using local search methods [41], and an improved version is designed later by Dreyer *et al.* [13]. Smith *et al.* propose a heuristic algorithm running in $O(n \log n)$ based on the generation and concentration of the full Steiner tree [40]. Bereta et al. propose a memetic algorithm to find optimal Steiner points [7]. These heuristic algorithms can generate an approximate solution in a polynomial time, suggesting that good handcrafted rules are helpful to solve the EST problems.

Learning to Solve Combinatorial Optimization Problem. Using machine learning to learn heuristic algorithms opens new horizons for solving combinatorial optimization problems. Vinyals *et al.* [43] propose a supervised learning method and design the pointer network model to learn a heuristic for computing the Delaunay triangulation and solving TSP. Later, Bello *et al.* [5] propose a reinforcement learning method to solve TSP; following this idea, Kool *et al.* [28] propose a deep reinforcement learning method based on the Transformer architecture. Motivated by the success on solving TSP, various combinatorial problems have been revisited by the machine learning community, e.g., the 3D bin packing problem [21], the minimum vertex cover problem [26], and the maximum cut problem [4,26]. More related works can be found in recent surveys [6,32].

Steiner Tree Over Graph vs Euclidean Steiner Tree. Being closely related to the EST problem, the Steiner tree over graph problem seeks to build the min-cost network using Steiner points from a given point set, implying the searching space is finite. Therefore, most of the existing techniques can be easily applied to the Steiner tree over graph problem. For example, Du *et al.* [14] propose a reinforcement learning method and Ahmed *et al.* [1] study the same problem through supervised learning methods. Another related one is the rectilinear Steiner minimum tree problem, where the network is required to be composed of horizontal and vertical lines, and therefore, the grid network is a natural choice for creating the searching space [9]. Unfortunately, such a simple method does not work well for the EST problem, as evidenced later in experiments.

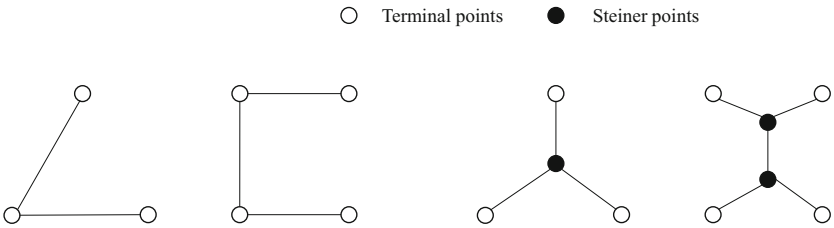

Fig. 1. The left two figures illustrate the minimal spanning trees. The right two figures illustrate the Steiner trees.

3 Problem Setting

Formally, the EST problem is defined as follows.

Definition 1 (The Euclidean Steiner tree (EST) problem). *Given a point set S (called terminal set) in a two-dimensional Euclidean space $\mathbb{R}^2$, construct a tree with the minimum Euclidean distance to connect S, where points not in S (called Steiner points) are allowed to use to construct the tree.*

An illustration of the comparison between minimal spanning trees and the Steiner tree is given in Fig. 1.

Concerning a distribution $\mathcal{D}$ over the input instance S, we wish to learn a function T that can compute a Steiner tree $T(S)$ over S with the minimum length. As a statistical learning problem, we seek to minimize the true error:

$$L(T) = \mathbb{E}_{S\sim\mathcal{D}}\Big[\operatorname{len}\big(T(S)\big)\Big]$$

where $\operatorname{len}(T(S))$ denotes the tree length under the Euclidean distance.

4 Deep-Steiner

In this section, we present a reinforcement learning method called Deep-Steiner to learn the desired function T.

4.1 Overall Structure

Notably, supposing that the optimal Steiner points have been identified, the optimal solution must be a minimum spanning tree over the Steiner points plus the terminal nodes. Therefore, the key part is to determine the Steiner points. The overall framework of Deep-Steiner is conceptually simple, as shown in Algorithm 1. Given the input instance S, Deep-Steiner decides the Steiner points in an iterative manner. In each iteration i, based on the current selected points $I_{i-1} \subseteq \mathbb{R}^2$ including both the terminal points and Steiner points, a new Steiner point is selected from a certain candidate set $C_i \subseteq \mathbb{R}^2$ by a policy $\pi : C_i \to \mathbb{R}^2$, with $I_0 = S$ being the initial input. In particular, the policy selects the best node

Algorithm 1. Deep-Steiner

Input: S and p_θ
Output: A Steiner tree associated with S

$I_0 = S, i = 1$
while $\text{len}(\text{MST}(I_i)) \leq \text{len}(\text{MST}(I_{i-1}))$ and $i \leq |S| - 2$ **do**
  Generate candidate set C_i based on I_{i-1}
  $v^* = \arg\max_{v \in C_i} p_\theta(v|I_{i-1})$, $I_i = I_{i-1} \cup \{v^*\}$, $i = i + 1$
end while
return $\text{MST}(I_{i-1})$

from the candidate set C_i according to a distribution $p_\theta(v|I_{i-1})$ over $v \in C_i$ (conditioned on I_{i-1}), where p_θ is parameterized by deep neural networks. In completing the framework, we first present methods for generating the candidate set (Sect. 4.2) and then present the design of p_θ (Sect. 4.3). Finally, we discuss training methods for computing the parameters θ involved in p_θ (Sect. 4.4).

4.2 Candidate Set C_i

We now present methods for constructing the candidate set C_i based on the current selected points I_{i-1}. To eschew searching the continuous space $\mathbb{R}^2$, one straightforward idea is to use the grid network, which has been widely adopted in existing methods [2,9,38]. Nevertheless, the grid pattern is not informed by the structure of the Steiner trees and therefore can lead to suboptimal performance, as shown in the experiments later. For a better candidate space, we are inspired by the following property of the Steiner trees.

Lemma 1. *[17] The Steiner points in the Steiner minimal tree must have a degree of three, and the three edges incident to such a point must form three 120° C angles.*

The above property suggests that the optimal Steiner points must be on the 120 °C-arc between two points in $\mathbb{R}^2$, which has greatly reduced the searching space. We denote such arcs as Steiner arcs.

Definition 2 (Steiner arc). *Given two points $a \in \mathbb{R}^2$ and $b \in \mathbb{R}^2$, the set $\text{Arc}_{a,b} = \{x : x \in \mathbb{R}^2, \angle axb = 120\}$ is the Steiner Arc induced by a and b.*

With the above concept, the initial space we acquire consists of all the Steiner arcs between the points in I_{i-1}. As such a space is still continuous, we divide the arc into k^* equal-size parts, for some $k^* \in \mathbb{Z}$, and select the endpoints to acquire an enumerable space. Formally, the resulted candidates associated with one arc $\text{Arc}_{a,b}$ are given as follows.

$$\text{Arc}_{a,b}^{k^*} = \left\{ x_1, ..., x_{k^*} : x_i \in \text{Arc}_{a,b}, |x_{i-1} - x_i| = |x_i - x_{i+1}|, x_0 = a, x_{k^*+1} = b, i \in [k^*] \right\}$$

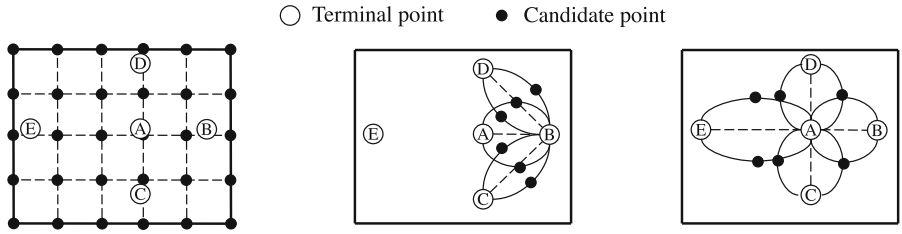

Fig. 2. The left figure illustrates the candidate points based on the grid network. The middle figure shows the candidate points generated based on the Steiner arcs between node B and its nearest neighbors; the right figure presents the candidate points generated by the MST-based methods.

Enumerating over all possible pairs in I_{i-1}, the total candidate points are in the order of $O(k^* \cdot |I_{i-1}|^2)$, which is still infeasible when being involved in training deep architectures (Sect. 4.3). Therefore, we adopt the following two methods to further reduce the searching space:

- **KNN-based candidate space.** Notice that the Steiner minimal tree has the optimality condition that each subgraph tree is also a minimum spanning tree over the involved points. Intuitively, for two points in I_{i-1} that are not local neighbors, their Steiner arc cannot include any Steiner points in the optimal solution to the input instance S. Therefore, for each point v in I_{i-1}, only the Steiner arcs between v and its nearest neighbors are selected. Let $N_v^{k'} \subseteq I_{i-1}$ be the set of the $k' \in \mathbb{Z}$ nearest neighbors of v in I_{i-1}. We have the following candidate space:

$$C_i = \bigcup_{v \in I_{i-1}} \bigcup_{u \in N_v^{k'}} \mathrm{Arc}_{v,u}^{k^*} \tag{1}$$

- **MST-based candidate space.** Following the same intuition of the KNN-based method, we now seek to eliminate unqualified Steiner arcs by using the minimum spanning tree $\mathrm{MST}(I_i)$ over I_i. That is, for each pair of the nodes in I_{i-1}, we only select the Steiner arc $\mathrm{Arc}_{u,v}$ when edge (u, v) appears in $\mathrm{MST}(I_i)$. Formally, we have the following space:

$$C_i = \bigcup_{(u,v) \in \mathrm{MST}(I_{i-1})} \mathrm{Arc}_{v,u}^{k^*} \tag{2}$$

One can easily see that the candidate space resulted from the above methods is linear in $|I_{i-1}|$ and determined by some hyperparameters k^* and k' that control the complexity-efficient trade-off. Such a trade-off will be experimentally studied later. An illustration of the candidate points generated by different methods is given in Fig. 2.

4.3 Policy π and Distribution p_θ

Given the candidate set C_i in each iteration of Algorithm 1, our policy computes one Steiner point based on a parameterized distribution $p_\theta(v|I_{i-1})$, with the hope that a higher value of $p_\theta(v|I_{i-1})$ translates the better quality of the final Steiner tree. To this end, the parameters θ will be trained using reinforcement learning (Sect. 4.4). In what follows, we present our design of $p_\theta(v|I_{i-1})$. The overall design of p_θ has two modules: the encode computes the hidden embedding of each point in C_i as well as the entire graph; the decoder translates the embeddings into a distribution over C_i.

Encoder. Given the candidate C_i and the currently selected points I_{i-1}, where each point is represented by its coordinates, we first compute the hidden representations following the framework proposed in [28] to capture useful latent relationships between the points in terms of building a Steiner minimal tree.

Let $X = C_i \cup I_{i-1} = \{x_1, ..., x_n\}$ be the entire point set. For each $x_j \in X$, the embedding is computed through the attention mechanism [42] composed of a sequence of $L \in \mathbb{Z}$ neural layers, where its d-dimensional hidden representation at layer $l \in [L]$ is denoted by $\mathbf{A}_{j,l} \in \mathbb{R}^{d\times 1}$. Initially, we have $\mathbf{A}_{j,0} = \mathbf{W}^{(0)} \cdot x_j + \mathbf{b}^{(0)}$ with $\mathbf{W}^{(0)} \in \mathbb{R}^{d\times 2}$, and $\mathbf{b}^{(0)} \in \mathbb{R}^{d\times 1}$ being learnable parameters. To obtain $\mathbf{A}_{j,l+1}$ from $\mathbf{A}_{j,l}$, we first use multi-head attention [42] to aggregate the information from the neighbors of x_j:

$$\mathrm{MHA}_{j,l+1}(\mathbf{A}_{1,l}, \mathbf{A}_{2,l}, ..., \mathbf{A}_{n,l}) = \sum_{m=1}^{M} \mathbf{W}_{m,l}^{(1)} \cdot \mathbf{A}_{j,m}^{'}$$

which is composed of $M \in \mathbb{Z}$ self-attentions $\mathbf{A}_{j,m}^{'}$ [42] computed by

$$\mathbf{A}_{j,m}^{'} = \sum_{k=1}^{n} \frac{\exp(\mathbf{q}_{j,m}^{\mathsf{T}} \mathbf{k}_{k,m}/\sqrt{d_s}) \cdot \mathbf{v}_{k,m}}{\sum_{k'=1}^{n} \exp(\mathbf{q}_{j,m}^{\mathsf{T}} \mathbf{k}_{k',m}/\sqrt{d_s})}$$

where $d_s = d/M$ is the normalizer, $\mathbf{q}_{j,m} \in \mathbb{R}^{d_s\times 1}$, $\mathbf{k}_{j,m} \in \mathbb{R}^{d_s\times 1}$, and $\mathbf{v}_{j,m} \in \mathbb{R}^{d_s\times 1}$ are hidden vectors obtained through linear transformation from $\mathbf{A}_{j,l}$:

$$\mathbf{q}_{j,m} = \mathbf{W}_{m,l+1}^{(2)} \cdot \mathbf{A}_{j,l}, \quad \mathbf{k}_{j,m} = \mathbf{W}_{m,l+1}^{(3)} \cdot \mathbf{A}_{j,l}, \quad \mathbf{v}_{j,m} = \mathbf{W}_{m,l+1}^{(4)} \cdot \mathbf{A}_{j,l}$$

with $\mathbf{W}_{m,l+1}^{(1)} \in \mathbb{R}^{d\times d_s}$, $\mathbf{W}_{m,l+1}^{(2)} \in \mathbb{R}^{d_s\times d}$, $\mathbf{W}_{m,l+1}^{(3)} \in \mathbb{R}^{d_s\times d}$, and $\mathbf{W}_{m,l+1}^{(4)} \in \mathbb{R}^{d_s\times d}$ being learnable parameters. With the transmission of MHA, the representation $\mathbf{A}_{i,l+1}$ of point x_j at the next layer $l+1$ is finally obtained through batch normalization and one feed-forward layer:

$$\hat{\mathbf{A}}_{j,l} = \mathrm{BN}_{l+1}\left(\mathbf{A}_{j,l} + \mathrm{MHA}_{j,l+1}(\mathbf{A}_{1,l}, \mathbf{A}_{2,l}, ..., \mathbf{A}_{n,l})\right)$$
$$\mathbf{A}_{j,l+1} = \mathrm{BN}_{l+1}\left(\hat{\mathbf{A}}_{j,l} + \mathrm{FF}_{l+1}(\hat{\mathbf{A}}_{j,l})\right)$$

where BN is the standard batch normalization process [23] and FF is the standard feed-forward network with ReLu activations [19] of dimension 512. Finally, the graph embedding $\mathbf{A}_G \in \mathbb{R}^d$ is calculated as the average sum of the final embeddings over I_{i-1} rather than over X, which is different from [28]:

$$\mathbf{A}_G = \frac{1}{|I_{i-1}|} \sum_{j:x_j \in I_{i-1}} \mathbf{A}_{j,L}$$

Decoder. With the final point embeddings $\mathbf{A}_{j,L}$ and the graph embedding $\mathbf{A}_G$, the encoder derives a parameterized distribution over C_i by combining the embeddings through exponential families:

$$p_\theta(x_j \in C_i | I_{i-1}) = \frac{\exp(\mathbf{q}^\mathsf{T} \cdot \mathbf{k}_j / \sqrt{d_s})}{\sum_{k:x_k \in X_i \cap C_i} \exp(\mathbf{q}^\mathsf{T} \cdot \mathbf{k}_k / \sqrt{d_s})}$$

where the hidden representation $\mathbf{q} \in \mathbb{R}^{d_s \times 1}$ and $\mathbf{k}_j \in \mathbb{R}^{d_s \times 1}$ are obtained via linear transformation based on the embeddings:

$$\mathbf{q} = \mathbf{W}^{(5)} \cdot \mathbf{A}_G \quad \text{and} \quad \mathbf{k}_j = \mathbf{W}^{(6)} \cdot \mathbf{A}_{j,L}$$

where $\mathbf{W}^{(5)} \in \mathbb{R}^{d_s \times d}$ and $\mathbf{W}^{(6)} \in \mathbb{R}^{d_s \times d}$ are learnable.

Parameter Space. In summary, the proposed model has three hyperparameters: $M \in \mathbb{Z}$ controlling the complexity of multi-head attention, $L \in \mathbb{Z}$ determining the number of layers in the encoder, and $d \in \mathbb{Z}$ specifying the hidden dimensions. On top of that, the distribution p_θ is parameterized by a collection of learnable matrices, including $\mathbf{W}^{(0)}$, $\mathbf{b}^{(0)}$, $\mathbf{W}^{(i)}_{m,l}$ ($i \in [1,4]$, $m \in [M]$ and $l \in [L]$), $\mathbf{W}^{(5)}$, and $\mathbf{W}^{(6)}$.

4.4 Training

We now present methods for computing the parameter using REINFORCE with baseline [28,46]. The framework is given in Algorithm 2. Each epoch consists of a sequence of batch training. In each batch training (line 4–20), given a batch size $B \in \mathbb{Z}$, we sample a collection of instances, and for each instance, we predict the Steiner trees based respectively on the current parameter θ and the best parameter θ_{bs}. With the predicted Steiner trees, we obtain a new parameter θ_{new} by maximizing

$$\sum_{i=1}^{B} (\text{len}(T_\theta(S_i)) - \text{len}(T_{\theta_{bs}}(S_i))) \cdot \bigtriangledown_\theta \log p_\theta(T_\theta(S_i)|S_i) \tag{3}$$

using the Adam algorithm [27], where $p_\theta(T_\theta(S_i)|S_i) = \prod_j p_\theta(I_j|I_{j-1})$ is likelihood of the prediction and I_j is the sequence of points used to construct $T_\theta(S_i)$. The above optimization implies that the model should produce a Steiner tree with small length and large likelihood. At the end of each epoch, the parameter will be updated only if it passes the paired t-test with $\alpha = 0.05$ on $10,000$ validation instances to examine whether or not the performance improvement is statistically significant [20]. Now the only part left is the process to generate predictions during the training process (line 6–17 in Algorithm 2), where we keep selecting Steiner points but use different stopping criteria for different training purposes. We consider three stopping criteria:

– **First-increment.** Similar to the inference process (Algorithm 1), we stop adding Steiner points if the resulted minimum spanning tree becomes larger instead of smaller. The rationale behind such a strategy is that parameter updating would become useless if the current tree is nowhere close to the optimal one.

Algorithm 2. REINFORCE with Rollout Baseline

Input: batch size B, significance α
Output: parameter θ_{bs}

initial parameter θ, $\theta_{bs} \leftarrow \theta$
for each epoch **do**
  for each batch training **do**
    Sample a batch of instance $\{S_1, ..., S_B\}$
    for $j \in [B]$ **do**
      $I_0 = S_j$, $i = 1$
      while stopping criterion is not met **do**
        Generate candidate set C_i based on I_{i-1}
        $v^{'} \sim p_\theta(v|I_{i-1})$, $I_i = I_{i-1} \cup \{v^{'}\}$, $i = i + 1$
      end while
      $T_\theta(S_i) = \mathrm{MST}(I_{i-1})$
      $I_0 = S_j$, $i = 1$
      while stopping criterion is not met **do**
        Generate candidate set C_i based on I_{i-1}
        $v^* = \arg\max_{v \in C_i} p_{\theta_{bs}}(v|I_{i-1})$, $I_i = I_{i-1} \cup \{v^*\}$, $i = i + 1$
      end while
      $T_{\theta_{bs}}(S_i) = \mathrm{MST}(I_{i-1})$
    end for
    $\bigtriangledown\mathcal{L} \leftarrow \sum_{i=1}^{B}(\mathrm{len}(T_\theta(S_i)) - \mathrm{len}(T_{\theta_{bs}}(S_i))) \cdot \bigtriangledown_\theta \log p_\theta(T_\theta(S_i)|S_i)$
    $\theta \leftarrow \mathrm{Adam}(\theta, \bigtriangledown\mathcal{L})$
  end for
  if t-test$(\theta, \theta_{bs}, \alpha)$ passes **then**
    $\theta_{bs} \leftarrow \theta$
  end if
end for

- **First-selection.** Under this stopping criterion, we stop adding Steiner points after selecting one Steiner point, which means that we focus on training the policy to select the nodes that are locally optimal. Since our inference process runs in a greedy manner, local optimality is necessary to achieve nontrivial performance compared to random methods.
- **All-selection.** Under such a method, we always select $|S| - 2$ Steiner points, which is motivated by the well-known fact that the Steiner minimal tree over $|S|$ points can have at most $|S| - 2$ Steiner points [17].

5 Experiment

In this section, we present our empirical studies.

5.1 Experiment Setting

Model Setting. We examine our method with different candidate generation methods (Sect. 4.2) and different stopping criteria (Sect. 4.4). Following the standard practice [37], we initialize the parameters using a pre-trained TSP-20 model

[28]. Balancing between quality and computational cost, the best hyperparameters are set as follows. $k^* = 9$ for arc partition; $k^{'} = 3$ for KNN-based candidate space; $d = 128, L = 5, M = 8$ for the attention model; the batch size B is 32.

Datasets. For synthetic dataset $(n, \mathcal{D})$, each instance is generated by randomly selecting n points following a certain distribution $\mathcal{D}$ over $\mathbb{R}^2$. Our experiments involve three synthetic datasets:

$$D_1 = (10, U(0,1)^2), D_2 = (20, U(0,1)^2), D_3 = (10, N(0.5, 0.2)^2),$$

where U is the uniform distribution and N is the Gaussian distribution. Such samples have been widely adopted in existing works [24,28]. In addition, we adopt the DEG-10 dataset containing a thousand instances with ten terminal points [48], which is a benchmark used in the 11th DIMACS Implementation Challenge [12].

Baseline Methods. The optimal solution is calculated by Geo-Steiner [44]. The second baseline outputs the minimum spanning tree of the input instance, which is a simple and effective approximation to the Steiner minimal tree. In addition, we include a highly effective heuristic method proposed by Bereta *et al.* [7], which is an iterative framework and can often produce near-optimal solutions provided with a sufficient number of iterations. Finally, two random methods are adopted as baselines. Rand-1 samples $r \in \mathbb{Z}$ random Steiner points from the distribution $\mathcal{D}$, where r is sampled from $[0, |S| - 2]$. Rand-2 is the same as Algorithm 1 except that the Steiner points are randomly selected from the candidate set; such a method is used to prove that our policy is indeed nontrivial.

Training and Testing. Our experiments were executed on Google Colab with NVIDIA Tesla V100. The training size is $10,240$. For each method, we examine its performance by comparing the predicted Steiner trees to the optimal solution over $10,000$ testing instances, i.e., $\frac{\text{len}(T_\theta(S)) - \text{len}(T_{\text{opt}}(S))}{\text{len}(T_{\text{opt}}(S))}$.

5.2 Result and Analysis

Overall Observations. Tables 1 and 2 show the performance in terms of effectiveness and efficiency. The results confirm that our method is non-trivially better than Rand-1 and minimum spanning trees, and furthermore, it is comparable to the best heuristic and can generate solutions close to the optimal ones. Compared to Berata's method, the main advantage of our method lies in time efficiency (as shown in Table 2), which suggests that methods based on reinforcement learning are promising in dealing with NP-hard combinatorial optimization problems with complex searching spaces.

On Candidate Generation Methods. According to Table 1, the proposed methods are clearly better than grid based methods in generating good candidate points. In addition, MST-based methods are better than the KNN-based on small graphs but not on large graphs. The main reason is that the candidate set generated by KNN-based methods is larger than that of the MST-based methods,

Table 1. Main results. Each cell shows the performance of one method on one dataset, together with the standard deviation. The missing data means that the training time of one epoch is more than 2.5 h, which is considered to be impractical.

Methods	D_1	D_2	D_3
Geo-Steiner	0	0	0
Bereta's method (10 iterations)	1.67 ± 0.02%	2.43 ± 0.01%	1.67 ± 0.02%
Bereta's method (20 iterations)	1.08 ± 0.01%	1.92 ± 0.01%	1.13 ± 0.02%
Bereta's method (50 iterations)	0.59 ± 0.01%	1.18 ± 0.01%	0.61 ± 0.01%
Minimum spanning tree	3.15 ± 0.03%	3.19 ± 0.01%	3.06 ± 0.02%
Rand-1	25.05 ± 3.31%	24.08 ± 2.06%	48.44 ± 19.1%
KNN+First-selection	2.41 ± 0.02%	2.13 ± 0.01%	2.62 ± 0.02%
KNN+First-increment	2.94 ± 0.02%		2.82 ± 0.02%
KNN+All-selection	3.02 ± 0.02%		2.92 ± 0.02%
KNN+Rand-2	3.64 ± 0.03%	3.24 ± 0.01%	3.45 ± 0.03%
MST+First-selection	1.39 ± 0.01%	2.46 ± 0.01%	1.77 ± 0.01%
MST+First-increment	2.30 ± 0.02%	3.01 ± 0.01%	2.07 ± 0.01%
MST+All-selection	2.95 ± 0.02%		2.92 ± 0.02%
MST+Rand-2	4.20 ± 0.04%	3.62 ± 0.02%	3.89 ± 0.04%
Grid+First-selection	2.98 ± 0.02%	3.16 ± 0.01%	2.88 ± 0.02%
Grid+First-increment	3.08 ± 0.02%	3.16 ± 0.01%	2.93 ± 0.02%
Grid+All-selection	3.07 ± 0.02%		2.91 ± 0.02%
Grid+Rand-2	3.31 ± 0.03%	3.21 ± 0.01%	3.23 ± 0.03%

Table 2. Running time. Each cell shows the average time cost to generate one prediction.

Methods	D_1	D_2	D_3
Geo-Steiner	211.2 ms	303.2 ms	205.2 ms
Bereta's method (10 iterations)	118.8 ms	331.7 ms	125.2 ms
Bereta's method (20 iterations)	310.0 ms	898.5 ms	315.4 ms
Bereta's method (50 iterations)	1021.5 ms	2217.1 ms	945.6 ms
Minimum spanning tree	1.6 ms	5.8 ms	1.7 ms
Rand-1	2.8 ms	10.7 ms	2.9 ms
KNN+First-selection	200.6 ms	285.1 ms	148.3 ms
KNN+Rand-2	9.4 ms	41.9 ms	8.7 ms
MST+First-selection	142.1 ms	254.9 ms	130.7 ms
MST+Rand-2	14.3 ms	22.3 ms	12.9 ms
Grid+First-selection	39.3 ms	72.1 ms	35.8 ms
Grid+Rand-2	7.0 ms	20.4 ms	7.0 ms

which means that, compared to the MST-based methods, KNN-based methods need more training epochs to get converged (especially on larger graphs) but can have a better performance once it is converged, for example, on small graphs. Indeed, we observed that both methods have almost converged on small graphs, while the loss of the KNN-based methods was still decreasing after 100 training epochs on large graphs.

Table 3. Generalization performance. Each cell shows the performance of one method generalized to one dataset, together with the standard deviation)

Methods	D_1	D_2	D_3	DEG
Model trained on D_1	1.39 ± 0.01%	2.72 ± 0.01%	1.63 ± 0.01%	1.47 ± 0.01%
Model trained on D_2	2.11 ± 0.02%	2.46 ± 0.01%	2.43 ± 0.05%	2.14 ± 0.02%
Model trained on D_3	1.75 ± 0.01%	2.81 ± 0.01%	1.77 ± 0.01%	1.81 ± 0.02%
Minimum spanning tree	3.15 ± 0.03%	3.19 ± 0.01%	3.06 ± 0.02%	3.29 ± 0.02%

On Stopping Criteria. According to Table 1, our proposed stopping criteria perform non-trivially better than the Rand-2, which confirms that our models indeed learn to improve the policy during training. An interesting observation is that First-selection has the best performance among the possible stopping criteria. One plausible reason is that in generating predictions in the training phase, compared to First-selection, First-increment and All-selection tend to construct larger trees on which the loss function is defined, which means that the corresponding models seek to learn a policy that can draw a Steiner tree, as opposed to the case of the First-selection where the model simply wants to learn how to select the best local Steiner point. For All-selection, it is less optimal also because it often forces the model to select some unnecessary candidates, as most of the Steiner minimal trees do not have $|S| - 2$ Steiner points.

Ability to Generalize Between Distributions. In order to investigate the generalization performance between distributions, we train the MST+First-selection on one dataset and evaluate its performance on different datasets. The result of this part is given in Table 3. The main observation is that even with distributions shifts, our method can still generate Steiner trees that are non-trivially better than the minimum spanning trees, which suggests that the embeddings we acquire do not heavily rely on the points distributions.

6 Conclusion

In this paper, we propose Deep-Steiner, a deep reinforcement learning method, to solve the EST problem. In particular, we design space discretization methods based on the unique properties of minimal Steiner trees, policy parameterization methods using attention techniques, and new training schemes with different stopping criteria. As evidenced by experiments, our method can generate decent solutions without incurring high computation and memory costs compared to traditional algorithms. One future work is to explore new representation methods in order to improve the performance on small graphs. In addition, it remains unknown how to overcome the memory issues in handling large instances, which is another interesting future work.

References

1. Ahmed, R., Turja, M.A., Sahneh, F.D., Ghosh, M., Hamm, K., Kobourov, S.: Computing Steiner trees using graph neural networks. arXiv preprint arXiv:2108.08368 (2021)
2. Ammar, A., Bennaceur, H., Châari, I., Koubâa, A., Alajlan, M.: Relaxed Dijkstra and A* with linear complexity for robot path planning problems in large-scale grid environments. Soft. Comput. **20**(10), 4149–4171 (2016). https://doi.org/10.1007/s00500-015-1750-1
3. Arora, S.: Polynomial time approximation schemes for Euclidean traveling salesman and other geometric problems. J. ACM (JACM) **45**(5), 753–782 (1998)
4. Barrett, T., Lvovsky, A., Clements, W., Foerster, J.: Exploratory combinatorial optimization with reinforcement learning. In: AAAI 2020–34th AAAI Conference on Artificial Intelligence, pp. 3243–3250 (2020)
5. Bello, I., Pham, H., Le, Q.V., Norouzi, M., Bengio, S.: Neural combinatorial optimization with reinforcement learning. ArXiv: abs/1611.09940 (2017)
6. Bengio, Y., Lodi, A., Prouvost, A.: Machine learning for combinatorial optimization: a methodological tour d'horizon. Eur. J. Oper. Res. **290**(2), 405–421 (2021)
7. Bereta, M.: Baldwin effect and Lamarckian evolution in a memetic algorithm for Euclidean Steiner tree problem. Memetic Comput. **11**(1), 35–52 (2019). https://doi.org/10.1007/s12293-018-0256-7
8. Caro, D.A.: Wireless Networks for Industrial Automation. ISA, Haryana (2005)
9. Chen, P.Y., et al.: A reinforcement learning agent for obstacle-avoiding rectilinear Steiner tree construction. In: Proceedings of the 2022 International Symposium on Physical Design, pp. 107–115 (2022)
10. Cheng, X., Du, D.Z., Wang, L., Xu, B.: Relay sensor placement in wireless sensor networks. Wirel. Netw. **14**(3), 347–355 (2008)
11. Deudon, M., Cournut, P., Lacoste, A., Adulyasak, Y., Rousseau, L.-M.: Learning heuristics for the TSP by policy gradient. In: van Hoeve, W.-J. (ed.) CPAIOR 2018. LNCS, vol. 10848, pp. 170–181. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-93031-2_12
12. DIMACS, t.D.S.F.o.I.S., Analysis, D.D., by the Institute for Computational, in Mathematics (ICERM), E.R.: 11th dimacs implementation challenge (2014). https://dimacs11.zib.de/downloads.html
13. Dreyer, D.R., Overton, M.L.: Two heuristics for the Euclidean Steiner tree problem. J. Glob. Optim. **13**(1), 95–106 (1998)
14. Du, H., Yan, Z., Xiang, Q., Zhan, Q.: Vulcan: Solving the Steiner tree problem with graph neural networks and deep reinforcement learning. arXiv preprint arXiv:2111.10810 (2021)
15. Fujiwara, T., Iida, N., Watanabe, T.: A hybrid wireless network enhanced with multihopping for emergency communications. In: 2004 IEEE International Conference on Communications (IEEE Cat. No. 04CH37577), vol. 7, pp. 4177–4181. IEEE (2004)
16. Garey, M.R., Graham, R.L., Johnson, D.S.: The complexity of computing Steiner minimal trees. SIAM J. Appl. Math. **32**(4), 835–859 (1977)
17. Gilbert, E.N., Pollak, H.O.: Steiner minimal trees. SIAM J. Appl. Math. **16**(1), 1–29 (1968)
18. Gong, H., Zhao, L., Wang, K., Wu, W., Wang, X.: A distributed algorithm to construct multicast trees in WSNs: an approximate Steiner tree approach. In: Proceedings of the 16th ACM International Symposium on Mobile Ad Hoc Networking and Computing, pp. 347–356 (2015)

19. Goodfellow, I., Bengio, Y., Courville, A.: Deep Learning. MIT Press, Cambridge (2016). https://www.deeplearningbook.org
20. Hsu, H., Lachenbruch, P.A.: Paired t test. Wiley StatsRef: statistics reference online (2014)
21. Hu, H., Zhang, X., Yan, X., Wang, L., Xu, Y.: Solving a new 3d bin packing problem with deep reinforcement learning method. arXiv preprint arXiv:1708.05930 (2017)
22. Hwang, F.K., Richards, D.S.: Steiner tree problems. Networks **22**(1), 55–89 (1992)
23. Ioffe, S., Szegedy, C.: Batch normalization: Accelerating deep network training by reducing internal covariate shift. In: International Conference on Machine Learning, pp. 448–456. PMLR (2015)
24. Joshi, C.K., Laurent, T., Bresson, X.: An efficient graph convolutional network technique for the travelling salesman problem. arXiv preprint arXiv:1906.01227 (2019)
25. Juhl, D., Warme, D.M., Winter, P., Zachariasen, M.: The Geosteiner software package for computing Steiner trees in the plane: an updated computational study. Math. Program. Comput. **10**(4), 487–532 (2018)
26. Khalil, E., Dai, H., Zhang, Y., Dilkina, B., Song, L.: Learning combinatorial optimization algorithms over graphs. In: Advances in Neural Information Processing Systems, vol. 30 (2017)
27. Kingma, D.P., Ba, J.: Adam: a method for stochastic optimization. In: ICLR (Poster) (2015)
28. Kool, W., van Hoof, H., Welling, M.: Attention, learn to solve routing problems! In: International Conference on Learning Representations (2018)
29. Li, D., Jia, X., Liu, H.: Energy efficient broadcast routing in static ad hoc wireless networks. IEEE Trans. Mob. Comput. **3**(2), 144–151 (2004)
30. Lin, G.H., Xue, G.: Steiner tree problem with minimum number of Steiner points and bounded edge-length. Inf. Process. Lett. **69**(2), 53–57 (1999)
31. Lu, X.X., Yang, S.W., Zheng, N.: Location-selection of wireless network based on restricted Steiner tree algorithm. Proc. Environ. Sci. **10**, 368–373 (2011)
32. Mazyavkina, N., Sviridov, S., Ivanov, S., Burnaev, E.: Reinforcement learning for combinatorial optimization: a survey. Comput. Oper. Res. **134**, 105400 (2021)
33. Min, M., Du, H., Jia, X., Huang, C.X., Huang, S.C.H., Wu, W.: Improving construction for connected dominating set with Steiner tree in wireless sensor networks. J. Glob. Optim. **35**(1), 111–119 (2006)
34. Nazari, M., Oroojlooy, A., Takáč, M., Snyder, L.V.: Reinforcement learning for solving the vehicle routing problem. In: Proceedings of the 32nd International Conference on Neural Information Processing Systems, pp. 9861–9871 (2018)
35. Obayiuwana, E., Falowo, O.E.: Network selection in heterogeneous wireless networks using multi-criteria decision-making algorithms: a review. Wirel. Netw. **23**(8), 2617–2649 (2017)
36. Peng, B., Wang, J., Zhang, Z.: A deep reinforcement learning algorithm using dynamic attention model for vehicle routing problems. In: Li, K., Li, W., Wang, H., Liu, Y. (eds.) ISICA 2019. CCIS, vol. 1205, pp. 636–650. Springer, Singapore (2020). https://doi.org/10.1007/978-981-15-5577-0_51
37. Qiu, X.P., Sun, T.X., Xu, Y.G., Shao, Y.F., Dai, N., Huang, X.J.: Pre-trained models for natural language processing: a survey. Sci. China Technol. Sci. **63**(10), 1872–1897 (2020). https://doi.org/10.1007/s11431-020-1647-3
38. Saeed, R.A., Recupero, D.R.: Path planning of a mobile robot in grid space using boundary node method. In: ICINCO (2), pp. 159–166 (2019)

39. Senel, F., Younis, M.: Relay node placement in structurally damaged wireless sensor networks via triangular Steiner tree approximation. Comput. Commun. **34**(16), 1932–1941 (2011)
40. Smith, J.M., Lee, D., Liebman, J.S.: An o (n log n) heuristic for Steiner minimal tree problems on the Euclidean metric. Networks **11**(1), 23–39 (1981)
41. Thompson, E.A.: The method of minimum evolution. Ann. Hum. Genet. **36** (1973)
42. Vaswani, A., et al.: Attention is all you need. In: Advances in Neural Information Processing Systems, pp. 5998–6008 (2017)
43. Vinyals, O., Fortunato, M., Jaitly, N.: Pointer networks. Adv. Neural Inf. Process. Sys. **28**, 2692–2700 (2015)
44. Warme, D., Winter, P., Zachariasen, M.: Geosteiner (2003)
45. Warme, D.M.: Spanning Trees in Hypergraphs with Applications to Steiner Trees. University of Virginia, Charlottesville (1998)
46. Williams, R.J.: Simple statistical gradient-following algorithms for connectionist reinforcement learning. Mach. Learn. **8**(3), 229–256 (1992)
47. Winter, P., Zachariasen, M.: Euclidean Steiner minimum trees: an improved exact algorithm. Netw.: Int. J. **30**(3), 149–166 (1997)
48. Wong, Y.C., Chu, C.: A scalable and accurate rectilinear Steiner minimal tree algorithm. In: 2008 IEEE International Symposium on VLSI Design, Automation and Test (VLSI-DAT), pp. 29–34. IEEE (2008)