NEUROPRI.M: AN ATTENTION-BASED MODEL FOR SOLVING NP-HARD SPANNING TREE PROBLEMS

Yuchen Shi
Department of Mathematical Sciences
University of Chinese Academy of Sciences
Beijing 100049, China
shiyuchen20@mails.ucas.ac.cn

Congying Han
Department of Mathematical Sciences
University of Chinese Academy of Sciences
Beijing 100049, China
hancy@ucas.ac.cn

Tiande Guo*
Department of Mathematical Sciences
University of Chinese Academy of Sciences
Beijing 100049, China
tdguo@ucas.ac.cn

June 13, 2023

ABSTRACT

Spanning tree problems with specialized constraints can be difficult to solve in real-world scenarios, often requiring intricate algorithmic design and exponential time. Recently, there has been growing interest in end-to-end deep neural networks for solving routing problems. However, such methods typically produce sequences of vertices, which makes it difficult to apply them to general combinatorial optimization problems where the solution set consists of edges, as in various spanning tree problems. In this paper, we propose NeuroPrim, a novel framework for solving various spanning tree problems by defining a Markov Decision Process (MDP) for general combinatorial optimization problems on graphs. Our approach reduces the action and state space using Prim’s algorithm and trains the resulting model using REINFORCE. We apply our framework to three difficult problems on Euclidean space: the Degree-constrained Minimum Spanning Tree (DCMST) problem, the Minimum Routing Cost Spanning Tree (MRCST) problem, and the Steiner Tree Problem in graphs (STP). Experimental results on literature instances demonstrate that our model outperforms strong heuristics and achieves small optimality gaps of up to 250 vertices. Additionally, we find that our model has strong generalization ability, with no significant degradation observed on problem instances as large as 1000. Our results suggest that our framework can be effective for solving a wide range of combinatorial optimization problems beyond spanning tree problems.

Keywords degree-constrained minimum spanning tree problem · minimum routing cost spanning tree problem · Steiner tree problem in graphs · Prim’s algorithm · reinforcement learning

1 Introduction

Network design problems are ubiquitous and arise in various practical applications. For instance, while designing a road network, one needs to ensure that the number of intersections does not exceed a certain limit, giving rise to a Degree-constrained Minimum Spanning Tree (DCMST) problem. Similarly, designing a communication network requires minimizing the average cost of communication between two users, which can be formulated as a Minimum Routing Cost Spanning Tree (MRCST) problem. In addition, minimizing the cost of opening facilities and communication network layouts while maintaining interconnectivity of facilities is critical for telecommunication network infrastructure.

*Corresponding author
This paper proposes NeuroPrim, a novel framework that utilizes the successful Transformer model to solve three
problems to optimality. Three variants of the spanning tree problem are considered in this paper. For each variant, an
outperforming strong heuristics.

In contrast to conventional algorithms that are designed based on expert knowledge, end-to-end models leveraging
deep neural networks have achieved notable success in tackling a wide range of combinatorial optimization problems
in graphs (STP). Although these problems all involve finding spanning trees in graphs, they have different constraints and objective functions, requiring heuristic algorithms tailored to their specific problem characteristics, without a uniform solution framework. Moreover, unlike the Minimum Spanning Tree (MST) problem, which can be solved in polynomial time by exact algorithms such as Prim’s algorithm [56], finding optimal solutions for these problems requires computation time exponential in problem size, making it impractical for real-world applications.

This paper proposes NeuroPrim, a novel framework that utilizes the successful Transformer model to solve three
challenging spanning tree problems in Euclidean space, namely the DCMST, MRCST and STP problems. The algorithm
is based on a Markov Decision Process (MDP) for general combinatorial optimization problems on graphs, which
significantly reduces the action and state spaces by maintaining the connectivity of partially feasible solutions, similar
to Prim’s algorithm. The framework is parameterized and trained with problem-related masks. Our goal is not to
outperform existing solvers; instead, we demonstrate that our approach can produce approximate solutions to various
sizes and variants of spanning tree problems quickly, with minimal modifications required after training. Our approach’s
effectiveness may provide insights into developing algorithms for problems with complex constraints and few heuristics.

Our work makes several significant contributions, including:

- We present a novel Markov Decision Process (MDP) framework for general combinatorial optimization
  problems on graphs. This framework allows all constructive algorithms for solving these problems to be viewed as greedy rollouts of some policy solved by the model.
- We propose NeuroPrim, a unified framework that leverages the MDP to solve problems that require decisions
  on both vertex and edge sets. Our framework is designed to be easily adaptable, requiring only minimal
  modifications to masks and rewards to solve different problems.
- To the best of our knowledge, we are the first to use a sequence-to-sequence model for solving complex
  spanning tree problems. We demonstrate the effectiveness of our approach by applying it to three challenging
  variants of the spanning tree problem on Euclidean space, achieving near-optimal solutions in a short time and
  outperforming strong heuristics.

2 Preliminaries

In this section, we describe the problems investigated and the techniques employed. Firstly, we present three vari-
ants of the spanning tree problem under investigation. Subsequently, we introduce the networks utilized for policy
parameterization.

2.1 Spanning tree problems

Given an undirected weighted graph \((V, G, E, w)\) with \(v(G)\) vertices, \(e(G)\) edges and \(w : E \rightarrow \mathbb{R}\) specifying
the weight of each edge, a spanning tree of \(G\) is a subgraph that is a tree which includes all vertices of \(G\). Among all
combinatorial optimization problems with spanning trees, the Minimum Spanning Tree (MST) problem is one of the
most typical and well-known one. The problem seeks to find a spanning tree with least total edge weights and can be
solved in time \(O(e(G) \log v(G))\) by Borůvka’s algorithm [10], Prim’s algorithm [56] and Kruskal’s algorithm [47].
However, with additional constraints or modifications to the objective function, it becomes more difficult to solve these
problems to optimality. Three variants of the spanning tree problem are considered in this paper. For each variant, an
example in Euclidean space is shown in Figure 1.
2.1.1 Degree-constrained Minimum Spanning Tree problem

Given an integer \( d \geq 2 \), the degree-constrained minimum spanning tree (DCMST) problem (also known as the minimum bounded degree spanning tree problem) involves finding a MST of maximum degree at most \( d \). The problem was first formulated by [52]. The NP-hardness can be shown by a reduction to the Hamilton Path Problem when \( d = 2 \) [33]. Several exact [5, 7, 16, 30], heuristic [8], metaheuristic [13, 14, 23, 25, 58, 67] algorithms and approximation schemes [4, 34, 66] have been proposed to tackle this problem. In Figure 1b, the path \( v_2v_3v_4v_1v_5 \) is the MST with maximum degree 2.

2.1.2 Minimum Routing Cost Spanning Tree problem

The minimum routing cost spanning tree (MRCST) problem (also known as the optimum distance spanning tree problem) aims to minimizing the sum of pairwise distances between vertices in the tree, i.e., finding a spanning tree of \( G \) that minimizes \( \sum_{(x, y) \in V(G) \times V(G), x < y} d_H(x, y) \), where \( d_H(x, y) \) is the total weights of the unique path between vertices \( x, y \in V(G) \) in the tree \( H \). This problem is first stated by [63] and then its NP-hardness is raised by [39]. Exact [11, 19, 31, 72, 81], heuristic [17, 51, 62, 64, 71], metaheuristic [61, 65, 70] algorithms and approximation schemes [79, 80] have been developed for this problem. In Figure 1c, the star graph with internal vertex \( v_1 \) and leaves \( v_2, v_3, v_4 \) and \( v_5 \) is the MRCST, and its routing cost is 4 times its total weight. (Note that after deleting any edge, the graph becomes two connected components with orders 1 and 4, so that the contribution of each edge to the routing cost is \( 1 \cdot 4 = 4 \).)

2.1.3 Steiner Tree Problem in graphs

Given a subset of vertices \( V' \subseteq V(G) \) called terminals, the Steiner tree problem in graphs (STP) asks for a tree in \( G \) that spans \( V' \) with minimum weights, i.e., finding a set \( S \subseteq V(G) \setminus V' \) that minimize the cost of the MST of \( G[V' \cup S] \), where \( G[V' \cup S] \) is the subgraph induced by \( V' \cup S \) and \( S \) is called Steiner points. The problem is NP-hard [42] and has two polynomially solvable variants: shortest path problem (\(|V'| = 2\)) and MST problem (\(|V'| = v(G)\)). Exact [22, 45, 50, 54], heuristic [28, 60, 69, 73], metaheuristic [5, 11, 29, 38, 41, 55, 57, 59] algorithms and approximation schemes [4, 15, 18] have been devised to address this problem. The most recent breakthrough lies in the 11th DIMACS (the Center for Discrete Mathematics and Theoretical Computer Science) Implementation Challenge and the PACE 2018 Parameterized Algorithms and Computational Experiments Challenge [9]. In Figure 1d, the star graph with internal vertex \( v_1 \) and leaves \( v_2, v_3, v_4 \) and \( v_5 \) is the Steiner tree connecting \( \{v_2, v_4, v_5\} \).

2.2 Transformer network

The Transformer [74] is a neural network architecture that was introduced in 2017 by Vaswani et al. It was designed to address some of the limitations of traditional recurrent neural networks (RNNs) and convolutional neural networks (CNNs) in processing sequential data. The Transformer has become a popular choice for various natural language processing (NLP) tasks, such as language translation, language modeling, and text classification.
The Transformer model architecture, as illustrated in Figure 2a, consists of two main components: the encoder and the decoder. The encoder takes the input sequence and produces a set of hidden states that represent the input sequence. The decoder takes the output of the encoder and produces the output sequence. Both the encoder and decoder consist of multiple layers of multi-head attention (MHA) and feed-forward networks (FFN). The MHA mechanism is the key component that allows the Transformer to model long-range dependencies in the input sequence.

Self-attention is a mechanism that allows the model to attend to different parts of the input sequence to compute a representation for each part. The self-attention mechanism employs query and key vectors to compute the attention weights, which determine the contribution of each element in the input sequence to the final output. The attention weights are computed by taking the dot product of the query and key vectors, dividing the result by the square root of the dimensionality of the key vectors, and normalizing the scores using the softmax function as depicted in Figure 2c. MHA extends self-attention by performing attention over multiple projections of the input sequence. In MHA, the input sequence is projected into multiple subspaces, and attention is performed separately on each subspace. This allows the model to attend to different aspects of the input sequence simultaneously, providing a richer representation of the input sequence. Specifically, the MHA block with residual connection (RC) and batch normalization (BN) can be defined as a parameterized function class $\text{MHA}(Q, K, V, M) = \text{BN}(Q + O)$, where

$$O = \prod_{h=1}^{H} \text{softmax} \left( M \odot (QW_h^q + 1_n b^q) (KW_h^k + 1_m b^k)^T / \sqrt{d} \right) (VW_h^o + 1_o b^o) W_h + 1_n b^o,$$

where $||$ is the matrix concatenation operator, $d$ is the embedding dimension of the model, $H$ is the number of parallel attention heads satisfying that $d$ is divisible by $H$, $\odot$ represents the element-wise product, $1_n = [1 \cdots 1]^T \in \mathbb{R}^{n \times 1}$, and $W_h^q, W_h^k, W_h^o \in \mathbb{R}^{d \times (d/H)}, b^q, b^k, b^o \in \mathbb{R}^{1 \times d}, W^o \in \mathbb{R}^{d \times d}, b^o \in \mathbb{R}^{1 \times d}$ and $h \in \{1, \ldots, H\}$ are trainable parameters. The MHA block is shown in Figure 2b.

The FFN with RC and BN can be similarly defined as $\text{FFN} : \mathbb{R}^{n \times d} \rightarrow \mathbb{R}^{n \times d}$, giving

$$\text{FFN}(X) = \text{BN}(X + (\text{ReLU}(XW_1 + 1_n b_1)W_2 + 1_n b_2)),$$

where $d_f$ is the dimension of hidden layer, and $W_1 \in \mathbb{R}^{d \times d_f}, b_1 \in \mathbb{R}^{1 \times d_f}, W_2 \in \mathbb{R}^{d_f \times d}$ and $b_2 \in \mathbb{R}^{1 \times d}$ are trainable parameters.

![Figure 2: The Transformer model architecture.](image)

### 3 Method

Our approach begins by defining an MDP for general combinatorial optimization problems on graphs. We then present the NeuroPrim algorithm and its constituent components for solving three variants of the spanning tree problem (i.e.,
We first give the specific forms of $G$ which takes as input the encoder embedding of graph $G$. The algorithm requires aggregated vertex, graph and edge features as input, which is obtained by a purely self-attentive method, the action space at $G$ spanning subgraphs of $G$.

3.3 Feature aggregation

$\pi$ 3.3 Feature aggregation

and Figure 3 illustrate the proposed NeuroPrim algorithm and a corresponding example, respectively.

$Q$ $S$

outputs the probability of available actions from state $s$.

$m$ 3.3 Feature aggregation

Drawing inspiration from [12], the policy is parameterized by vanilla Transformer model [74] as $\pi_\theta(A_t \mid G, S_t)$, which takes as input the encoder embedding of graph $G$ and the concatenated decoder output representing $S_t$, and outputs the probability of available actions from state $S_t$. The goal is to train $\pi_\theta$ to maximize the expected flat partial return $J(\theta) = E_{\pi_\theta} [R_T]$, where $E_{\pi_\theta} [\cdot]$ is taken over all trajectories under $\pi_\theta$, each with a generation probability of $\prod_{t=0}^{T-1} \pi_\theta(A_t \mid G, S_t)$. The model is trained by REINFORCE using the POMO baseline with sampling. Algorithm 1 and Figure [5] illustrate the proposed NeuroPrim algorithm and a corresponding example, respectively.

3.3 Feature aggregation

The algorithm requires aggregated vertex, graph and edge features as input, which is obtained by a purely self-attentive based encoder.
The encoder first computes the initial vertex embeddings $H^0_v$ through a trainable linear projection with parameters $W^v \in \mathbb{R}^{2 \times d_v}$ and $b^v \in \mathbb{R}^{1 \times d_v}$, i.e., $H^0_v = XW^v + 1_{v(G)}b^v$, where $X$ is the 2-dimensional coordinates of instance $G$ and $d_v$ is the dimension of vector features. The vertex features $h^v_l$ are then calculated by

$$H^v_l = \text{FFN} \left( \text{MHA} \left( H^v_{l-1}, H^v_{l-1}, H^v_{l-1}, J_{v(G)} \right) \right),$$

$$[h^v_1 \ldots h^v_{L_v(G)}] = (H^v_{L_v(G)})^T,$$

where $l \in \{1, \ldots, L_v(G)\}$ is the encoder layer index. Vertices of different classes (e.g., terminal and non-terminal vertices) do not share parameters.

The graph feature $h^g$ is the aggregation of all vertex features, i.e., $h^g = W^g \left( \sum_{i=1}^{v(G)} h^v_i / v(G) \right) + b^g$, where $W^g \in \mathbb{R}^{2 \times d_e}$ and $b^g \in \mathbb{R}^{d_e}$ are trainable parameters.

One popular approach to obtain edge features is to use a function that takes the feature vectors of the two nodes connected by an edge $(i, j)$ as input and outputs a corresponding edge feature $h^e_{ij}$. One such function is given by the equation:

$$h^e_{ij} = \left[ \frac{h^v_i - h^v_j}{\max(h^v_i, h^v_j)} \right],$$

Figure 3: The NeuroPrim algorithm solves the DCMST $(d = 2)$ problem by generating a sequence of actions that form the optimal solution $(v_2, v_3, v_4, v_1, v_5)$. The dashed arrows illustrate the state transitions in the Markov Decision Process (MDP), while the solid arrows indicate the flow of parameterized state and action messages. The blue vertices and edges represent the current state, starting from the initial state $K_0$. The encoder processes the graph coordinates to obtain vertex, edge, and graph embeddings. At each time step, the decoder takes the previous actions and graph features as inputs, and produces masked attention weights to determine the next action. The mask is computed through a Hadamard product of four masks, with the purple element indicating the edge index that can be selected.
where $h_v^i$ and $h_j^i$ are the feature vectors of nodes $i$ and $j$, respectively. The first component of the edge feature vector corresponds to the difference between the feature vectors of the two nodes, which characterizes the relative discrepancy or distance their attributes and is informative for capturing the directionality of the relationship between them. The second component is the maximum of the feature vectors of the two nodes, which represents the shared information between the two nodes and reflects the similarity or dissimilarity between the features of the two nodes. The edge features are subsequently linearly transformed to $H_e$ for later use in decoding:

$$H_e = [h_{11}^e \cdots h_{1n}^e \cdots h_{n1}^e \cdots h_{nn}^e]^T W_e + 1_{v(G)} b^e,$$

where $W_e \in \mathbb{R}^{2d_e \times d_e}$ and $b^e \in \mathbb{R}^{1 \times d_e}$ are trainable parameters, and $d_e$ is the dimension of edge features.

### 3.4 Policy parameterization

The parameterization of the policy relies on an autoregressive decoder, which uses the information from previous time steps to generate the value of the current time step. Let $A_{0:t-1} = (v_0, u_1 v_1, \ldots, u_{t-1} v_{t-1})$ denotes the action list and also the partial solution before time step $t$, where $v$ is a randomly selected vertices to be spanned. We take the graph feature together with the feature representation of $A_{0:t-1}$ as input to the decoder at time $t$:

$$h_{t,0} = \text{MaxPool} \left( \left[ h^v_{u_{t-1} v_{t-1}} \right] \right),$$

where $u_0 = v_0$ and MaxPool is a downsampling operator with kernel size 2.

The hidden state $h_{t, dec}$ is obtained by applying $L^{\text{dec}}$ MHA blocks over the partial solution:

$$H_{t,0} = [h_{1,0} \cdots h_{t,0}]^T,$$

$$h_{t,l} = \text{MHA} (h_{t,l-1}, H_{t,l-1}, H_{t,l-1}, J_{1:l-1}),$$

where $l \in \{1, \ldots L^{\text{dec}}\}$ is the encoder decoder index and $J_{1:l-1}$ is a matrix of ones.

The probability distribution of actions in state $S_t$ is finally computed by $h_{t,0}^p = h_{t, dec}$ through $L^{\text{dec}}$ encoder-decoder MHA blocks and a single-head attention with only query and key:

$$h_{t,l}^p = \text{MHA}(h_{t,l-1}^p, H_e, H_e, M_l),$$

$$\pi_{\theta}(A_t \mid G, S_t) = \text{softmax} \left( C \tanh \left( M_l \odot (h_{t,dec}^p W_1 + 1 b_1)(H_e W_2 + 1 b_2)^T / \sqrt{d} \right) \right).$$

where $l \in \{1, \ldots L^{\text{dec}}\}$, $C = 10$ is a clipping threshold, $M_l \in \{0, 1\}^{v(G) \times v(G)}$ is determined jointly by $A_{S_t}$ and the problem related constraints, and $W_1, W_2 \in \mathbb{R}^{d_e \times d_e}$ and $b_1, b_2 \in \mathbb{R}^{1 \times d_e}$ are trainable parameters. Specifically, the mask $M_l$ is determined by four masks: edge mask $M^e_l$, unvisited vertex mask $M^v_t$, visited vertex mask $M^u_t$ and constraint related mask $M^c_t$ through Hadamard product, which guarantee that the state has no loop, is a tree, and satisfies the problem constraints. The masks are asymmetric and are generated by treating actions as directed edges, which reflects the form of the reduced action space $A_t$. For example, the unvisited vertex mask in Figure 3 indicates that the end of the selected action must be a vertex that is not in the current state, i.e., $v_2$, which corresponds to the second column of the mask. The specified form of the masks for the DCMST ($d = 2$) problem is given in Table 1. The MRCST and STP problems share the first three masks with DCMST, while $M^c_t = I$ for any time step $t$, since these two problems do not have any constraints on the selection of edges.

### Table 1: The masks for the DCMST ($d = 2$) problem, where $J$ and $O$ represent matrices of all ones and zeros respectively, $I$ is the identity matrix, and $e_{ij}$ denotes the matrix with 1 in the $i$th row and $j$th column row only and all other matrices are 0. The mask at time step $t$ is obtained by calculating $M_t = M^e_t \odot M^v_t \odot M^u_t \odot M^c_t$. All these matrices are square matrices of order $v(G)$.

| Time Step | $M^e_t$ | $M^v_t$ | $M^u_t$ | $M^c_t$ |
|-----------|---------|---------|---------|---------|
| $t = 0$   | $J - I$ | $J$     | $O$     | $J$     |
| $t = 1$   | $J - I$ | $J - \sum_i e_{iv_0}$ | $O + \sum_j e_{vu_j}$ | $J$     |
| $t \in \{2, \ldots, T\}$ | $J - I$ | $M_{t-1}^e - \sum_i e_{iv_{t-1}}$ | $M_{t-1}^v + \sum_j e_{vu_{t-1}j}$ | $\max (O, M_{t-1}^c - \sum_i e_{iv_{t-1}} - \sum_j e_{vu_{t-1}j})$ |
3.5 Policy optimization

After sampling $N$ solutions with $\pi_\theta$, we obtain a set of trajectories $\{(S^n_{t-1}, A^n_{t-1}, R^n_t)_{t=1}^{T_n}\}_{n=1}^N$, where $T_n$ is the final time step of the $n$th trajectory. The gradient of the expected flat partial return $J(\theta)$ can be approximated by the policy gradient theorem \cite{68} as follows

$$\nabla J(\theta) \approx \frac{1}{N} \sum_{n=1}^N \left( R^n_{T_n} - b \right) \nabla \log \prod_{t=1}^{T_n-1} \pi_\theta(A^n_t \mid G, S^n_t),$$

(14)

where $b = \frac{1}{N} \sum_{n=1}^N R^n_{T_n,n}$ is the POMO baseline sampled from $N$ trajectories decoded from randomly selected vertices and the start time step $t$ is set to 1 because the first action is a randomly selected vertex and therefore not considered in the policy. We use Adam \cite{44} as the optimizer. The training procedure is described in Algorithm 2.

**Algorithm 2** REINFORCE with sampled POMO

**Input:** number of epochs $E$, batch size $B$, trajectories sampled per instance $N$

1: Initialize $\theta$
2: for epoch $= 1, \ldots, E$ do
3:   for $i = 1, \ldots, B$ do
4:     $G_i \leftarrow \text{RandomInstance}()$
5:   for $n = 1, \ldots, N$ do
6:     $A^n_0 \leftarrow \text{RandomSelect}(V(G_i))$
7:     $(S^n_{t-1}, A^n_{t-1}, R^n_t)_{t=2}^{T_n} \leftarrow \text{Rollout}(G_i, \pi_\theta)$
8:   end for
9:   $b \leftarrow \frac{1}{N} \sum_{n=1}^N R^n_{T_n,n}$
10: $\nabla J \leftarrow \frac{1}{N} \sum_{n=1}^N (R^n_{T_n,n} - b) \nabla \log (\prod_{t=1}^{T_n-1} \pi_\theta(A_{t,n} \mid G_i, S_{t,n}))$
11: end for
12: $\theta \leftarrow \text{Adam}(\theta, \nabla J)$
13: end for

4 Experiments

We concentrate our study on three variations of the spanning tree problem, specifically the DCMST ($d = 2$), MRCST, and STP problems. Among these, we focus solely on the DCMST problem with a degree constraint of 2 due to its perceived difficulty in the literature. For each problem, we modify the masks and rewards and train the model on instances of size 50 generated from $U_{[0,1]} \times [0,1]$ following the methodology in \cite{46}. Both training and testing are executed on a GeForce RTX 3090 GPU. We determine the gap by utilizing the formula $(\text{Obj.} / \text{BKS} - 1) \times 100\%$, where Obj. signifies the objective obtained by a specific algorithm and BKS represents the best known solution. Our code in Python is publicly available on

https://github.com/CarlossShi/neuroprim

**Hyperparameters** We set the dimensions of vector, edge features and hidden layer of FFN to $d_v = 256$, $d_e = 16$ and $d_f = 512$, respectively. We use $H = 8$ heads, $L^\text{enc} = 3$ layers in the encoder and $L^\text{dec} = 1$ in the decoder. The models are trained for $E = 100$ epochs, each with $B = 2500$ batches of approximately 512 instances generated by random seed 0, where the exact number of instances satisfies a product of graph size $\nu(G)$ with the number of trajectories $N$ sampled per graph. We use a fixed learning rate $\eta = 10^{-4}$.

**Decoding strategy** We present two decoding strategies denoted as NeuroPrim (g) and NeuroPrim (s), where the former generates the output greedily and the latter uses sampling based on the masked attention weights to produce the result.

**Run times** For the DCMST ($d = 2$), MRCST and STP problems, the training time for NeuroPrim is 27 m, 47 m and 14 m respectively. The longer time for training the MRCST model is due to the inability to parallelize the reward computation, which is attributed to the specific optimization objective of the problem. Unlike these two problems, the final time step $T$ may differ in STP because the number of terminals varies from instance to instance. Thus, for each batch of training data, we sample the number of terminals from $U(2, \nu(G))$, and this is why its training time is about
half that of the DCMST problem. For testing, we find that the time required for NeuroPrim to solve these problems is linear in $T$. A better solution can be obtained by randomly selecting the initial vertices for decoding and sampling multiple solutions and taking the best.

4.1 Results on benchmark datasets

We report the results of benchmark tests on Euclidean space for each problem using our algorithm. Prior to testing, we preprocessed the data by scaling and translating all coordinates to fit within the range of $[0, 1]^2$. For the NeuroPrim(g) decoding method, we determined the best outcome by traversing $|A_{K_0}|$ initial vertices and considered it to be a deterministic algorithm. In the case of the NeuroPrim(s) decoding approach, we assessed the cost by sampling $20|A_{K_0}|$ initial vertices from the action distribution generated by the policy, and we report the mean, standard deviation and best result.

Degree-constrained Minimum Spanning Tree problem  We present the computational results on CRD dataset, which contains 10 Euclidean instances each of sizes 30, 50, 70 and 100. By reducing this problem to a TSP problem, we obtained BKS and approximate solutions using the Concorde and farthest insertion algorithm respectively. We take the farthest insertion algorithm into account because it is an effective constructive algorithm that adds one vertex per step and is therefore very similar to the execution of NeuroPrim.

As presented in Table 2, we find that NeuroPrim outperforms the furthest insertion algorithm in terms of optimality gap on all instances. If enough trajectories are sampled, NeuroPrim (s) is able to find optimal solutions for some small-scale problems.

Table 2: Results for DCMST ($d = 2$) problem on CRD instances.

| Ins. | $v(G)$ | BKS | Farthest Insertion | NeuroPrim (g) | NeuroPrim (s) |
|------|--------|-----|-------------------|---------------|---------------|
|      |        |     | Ins. | Obj. | Gap | Obj. | Gap | Mean Obj. | Obj. | Gap | Time |
| crd300 | 30 | 3822 | 4231 | 10.69% | 3834 | 0.31% | 0.2s | 3968±164 | 3822 | 0.00% | 2.4s |
| crd301 | 30 | 3616 | 3822 | 4.08% | 3663 | 1.28% | 0.2s | 3759±71 | 3622 | 0.14% | 2.3s |
| crd302 | 30 | 4221 | 4450 | 5.42% | 4256 | 0.83% | 0.1s | 4375±89 | 4221 | 0.00% | 2.2s |
| crd303 | 30 | 4233 | 4440 | 5.42% | 4256 | 0.83% | 0.1s | 4375±89 | 4221 | 0.00% | 2.2s |

3 https://h3turing.cs.hbg.psu.edu/~bui/data/bui-zrncic-gecco2006.html
4 https://www.math.uwaterloo.ca/tsp/concorde.html
5 https://github.com/wouterkool/attention-learn-to-route/blob/master/problems/tsp/tsp_baseline.py
Minimum Routing Cost Spanning Tree problem  We consider 21 instances of sizes 50, 100, 250 obtained from OR-Library\(^6\) which were originally benchmark for Euclidean Steiner problem, but were first used by [40] for testing on the MRCST problem. Since solving MRCST exactly is extremely expensive (e.g., [81] took a day to reduce the gap of a 100-size instance to about 10%), we use the experimental results reported in [61] as BKS. We implement the heuristic proposed by Campos [17] using Python and report the results. Campos’s algorithm is chosen because it is essentially a variant of Prim’s algorithm, with the difference that the metric for selecting edges is changed.

The results are shown in Table 3. We found that NeuroPrim significantly outperforms Campos’s algorithm for instances of size 100 or less. For problems as large as 250, sampling is effective in reducing the optimization gap of the solution.

| Ins. | \(v(G)\) | BKS | Campos | NeuroPrim (g) | NeuroPrim (s) |
|------|-----------|-----|--------|---------------|---------------|
|      |           |     | Obj.   | Gap | Obj.   | Gap | Time | Mean Obj. | Obj.   | Gap | Time |
| e50.1 | 50        | 984 | 1028   | 4.48% | 987   | 0.34% | 0.4s | 988 ±1    | 987   | 0.31% | 6.9s |
| e50.2 | 50        | 901 | 953    | 5.78% | 914   | 1.43% | 0.2s | 915 ±1    | 914   | 1.42% | 5.2s |
| e50.3 | 50        | 888 | 935    | 5.27% | 892   | 0.44% | 0.2s | 893 ±1    | 892   | 0.37% | 5.2s |
| e50.4 | 50        | 777 | 846    | 8.96% | 786   | 1.16% | 0.2s | 787 ±1    | 784   | 0.96% | 5.2s |
| e50.5 | 50        | 848 | 910    | 7.35% | 850   | 0.26% | 0.2s | 851 ±0    | 850   | 0.24% | 5.3s |
| e50.6 | 50        | 818 | 869    | 6.22% | 834   | 1.96% | 0.2s | 833 ±1    | 831   | 1.57% | 5.3s |
| e50.7 | 50        | 866 | 907    | 4.79% | 882   | 1.91% | 0.2s | 883 ±1    | 879   | 1.53% | 5.2s |
| e100.1 | 100      | 3507| 3702   | 5.57% | 3551  | 1.26% | 0.8s | 3559 ±6   | 3542  | 1.01% | 19.7s |
| e100.2 | 100      | 3308| 3497   | 5.70% | 3348  | 1.21% | 0.8s | 3355 ±7   | 3341  | 1.01% | 16.8s |
| e100.3 | 100      | 3566| 3736   | 4.76% | 3624  | 1.61% | 0.8s | 3625 ±4   | 3608  | 1.16% | 16.3s |
| e100.4 | 100      | 3448| 3643   | 5.64% | 3504  | 1.63% | 0.8s | 3511 ±8   | 3490  | 1.22% | 16.4s |
| e100.5 | 100      | 3637| 3857   | 6.05% | 3761  | 3.41% | 0.8s | 3769 ±5   | 3753  | 3.18% | 16.3s |
| e100.6 | 100      | 3436| 3601   | 4.78% | 3480  | 1.26% | 0.8s | 3489 ±6   | 3477  | 1.18% | 16.2s |
| e100.7 | 100      | 3704| 3914   | 5.69% | 3739  | 0.97% | 0.8s | 3747 ±6   | 3737  | 0.92% | 16.2s |
| e250.1 | 250      | 22088| 23324 | 5.60% | 22758 | 3.03% | 15.2s | 22905 ±182 | 22651 | 2.55% | 5.6m |
| e250.2 | 250      | 22771| 24031 | 5.53% | 23307 | 2.35% | 14.9s | 23379 ±37  | 23249 | 2.10% | 5.5m |
| e250.3 | 250      | 21871| 23241 | 6.26% | 22298 | 1.95% | 14.7s | 22415 ±59  | 22253 | 1.74% | 5.3m |
| e250.4 | 250      | 23423| 24660 | 5.28% | 23899 | 2.03% | 15.0s | 24019 ±89  | 23788 | 1.56% | 5.0m |
| e250.5 | 250      | 22378| 23517 | 5.09% | 22915 | 2.40% | 14.7s | 23075 ±80  | 22855 | 2.13% | 5.7m |
| e250.6 | 250      | 22285| 23282 | 4.47% | 22619 | 1.50% | 14.9s | 22790 ±80  | 22584 | 1.34% | 5.8m |
| e250.7 | 250      | 22909| 24017 | 4.84% | 23477 | 2.48% | 14.9s | 23533 ±57  | 23367 | 2.00% | 5.8m |

Mean | 5.62% | 1.65% | 1.40%
approximate solution by simply applying the Kruskal’s algorithm implemented by SciPy on the induced subgraph of all the terminals, as there is little difference in the performance of the algorithms on this problem.

Based on the result presented in Table 4, it is evident that NeuroPrim struggles to effectively learn the process of identifying Steiner points to narrow the optimization gap. The performance gap may be attributed to the nature of the problem and the under-sampling of instances that exceed the size of the training set when the stochastic algorithm is executed. Despite this limitation, the algorithm’s overall performance is comparable to that of Kruskal’s algorithm.

### Table 4: Results for STP on P4E instances.

| Ins.  | $v(G)$ | $|V'|$ | BKS | Kruskal Obj. | Gap | Obj. | Gap | Time | NeuroPrim (g) Obj. | Gap | Time | NeuroPrim (s) Obj. | Gap | Time |
|-------|--------|------|-----|-------------|-----|------|-----|------|-------------------|-----|------|-------------------|-----|------|
| p455  | 100    | 5    | 1138| 1166 | 2.46% | 1166 | 2.46% | 0.1s | 1166±0           | 1166 | 2.46% | 0.7s              |      |       |
| p456  | 100    | 5    | 1228| 1239 | 0.90% | 1239 | 0.90% | 0.0s | 1239±0           | 1239 | 0.90% | 0.5s              |      |       |
| p457  | 100    | 10   | 1609| 1642 | 2.05% | 1642 | 2.05% | 0.1s | 1642±1           | 1642 | 2.05% | 1.0s              |      |       |
| p458  | 100    | 10   | 1868| 1868 | 0.00% | 1868 | 0.00% | 0.1s | 1884±17          | 1868 | 0.00% | 0.9s              |      |       |
| p459  | 100    | 20   | 2345| 2348 | 0.13% | 2348 | 0.13% | 0.4s | 2365±13          | 2348 | 0.13% | 2.1s              |      |       |
| p460  | 100    | 20   | 2959| 3015 | 1.89% | 3015 | 1.89% | 0.1s | 3030±14          | 3010 | 1.72% | 1.8s              |      |       |
| p461  | 100    | 50   | 4474| 4492 | 0.40% | 4492 | 0.40% | 0.2s | 4528±17          | 4493 | 0.42% | 5.1s              |      |       |
| p462  | 200    | 10   | 1510| 1545 | 2.32% | 1545 | 2.32% | 0.1s | 1545±0           | 1545 | 2.32% | 1.4s              |      |       |
| p463  | 200    | 20   | 2545| 2574 | 1.14% | 2574 | 1.14% | 0.1s | 2593±13          | 2574 | 1.14% | 3.4s              |      |       |
| p464  | 200    | 40   | 3853| 3880 | 0.70% | 3880 | 0.70% | 0.4s | 3972±504         | 3880 | 0.70% | 9.7s              |      |       |
| p465  | 200    | 100  | 6234| 6253 | 0.30% | 6253 | 0.30% | 1.7s | 6527±143         | 6341 | 1.72% | 35.0s             |      |       |

### 4.2 Generalization on larger instances

We further test NeuroPrim for generalizability at problem sizes on instances as large as 1000. We randomly generate test instances of size $n = 50, 100, 200, 500, 1000$ on $U_{[0,1]}$, with $10000/n$ of each kind, sample 10 results of greedy decoding each, and average the best results obtained by instance size. The BKS for the DCMST ($d = 2$) and STP problems are obtained by the Concorde solver and the exact solving framework proposed by Leitner [49], respectively. For the MRCST problem, we use the Campos’s algorithm to obtain the BKS due to the difficulty of solving the exact solutions. From the results in Table 5, we can see that the optimality gap obtained by NeuroPrim increases gradually with the problem size. We obtain gaps of 13.44% and 1.44% for the DCMST ($d = 2$) and STP problems of size 1000, respectively, while the MRCST problem outperforms the heuristic results in all tested sizes.

### Table 5: Generalization ability of NeuroPrim on random instances.

| Problem  | 50  | 100 | 200 | 500 | 1000 |
|----------|-----|-----|-----|-----|------|
| DCMST ($d = 2$) | 1.67% | 3.89% | 7.80% | 11.74% | 13.44% |
| MRCST | -4.86% | -4.08% | -3.20% | -1.90% | -1.27% |
| STP ($|V'| = v(G)/2$) | 0.82% | 0.91% | 1.08% | 1.39% | 1.44% |

### 4.3 Sensitivity Analysis

This section presents a one-at-a-time sensitivity analysis of six hyperparameters: random seed, learning rate $\eta$, model dimension $d_e$ and $d_f$, number of heads used in MHA blocks, number of encoder layers and number of decoder layers. The remaining parameters are kept constant as specified above, except for the batch size $B$, which is set to 1000. Specifically, each hyperparameter is individually adjusted while holding the remaining parameters constant. The aim is to observe the resulting training losses and performance on randomly generated test instances for each problem. Figures 4, 5, and 6 illustrate the loss and cost results obtained from training in DCMST ($d = 2$), MRCST and STP problems, respectively, plotted against the model epoch.

Our findings indicate that the learning rate is a crucial hyperparameter for achieving model convergence and optimal performance in all problems. A large learning rate can lead to a failure to converge; hence, a straightforward and

https://github.com/scipy/scipy
effective choice is to set the learning rate to 0.0001. Although increasing the model dimension, the number of heads, and the number of encoding layers typically improves the trained algorithm’s performance, the improvement is usually not significant, except in the MRCST problem. Here, setting the number of heads to 16 may result in a local optimum, and thus, a simpler and more stable option is to set the number of heads to 8. Additionally, the number of decoding layers typically enhances the algorithm’s performance, especially in the DCMST ($d = 2$) problem. Our analysis shows that the choice of random seeds has minimal effect on the effectiveness of model training.
4.4 Example solutions

As shown in Figure 7, we choose an instance of size 50 and visualize solutions in three problems, i.e., DCMST \((d = 2)\), MRCST, STP \(|V'| = 25\), using greedy decoding strategy. For DCMST, the edges in the solution set are not interleaved and strictly adhere to the degree constraint of the vertices. In MRCST, the set of edges radiates outwards from the center, which is consistent with the goal of minimizing the average pairwise distance between two vertices. As for STP, the model is able to distinguish between terminals and ordinary vertices, and constructs the solution set step by step according to the distribution of terminals. The results show that NeuroPrim is able to learn new algorithms efficiently for general combinatorial optimization problems on graphs with complex constraints and few heuristics.

![Figure 6: Sensitivity analysis of hyperparameters on STP](image)

**Figure 6:** Sensitivity analysis of hyperparameters on STP

**Figure 7:** Example solutions of an instance of size 50 for DCMST \((d = 2)\), MRCST, STP \(|V'| = 25\). The terminals are represented by blue nodes, and the edges in solutions by black arrows. The vertex at the beginning of the solution are highlighted in red.

5 Conclusion

In this paper, we introduced a unified framework called NeuroPrim for solving various spanning tree problems on Euclidean space, by defining the MDP for general combinatorial optimization problems on graphs. Our framework achieves near-optimal results in a short time for the DCMST, MRCST and STP problems, without requiring extensive...
problem-related knowledge. Moreover, it generalizes well to larger instances. Notably, our approach can be extended to solve a wider range of combinatorial optimization problems beyond spanning tree problems.

Looking ahead, there is a need to explore more general methods for solving combinatorial optimization problems. While there are existing open source solution frameworks such as [37, 82], they often rely on designing specific algorithm components for individual problems. Additionally, scaling up to larger problems poses a challenge, as the attention matrix may become too large to store in memory. A possible solution to this is to use local attention within a neighborhood, rather than global attention, which may unlock further potential for the efficiency of our algorithm.

Acknowledgements

This research was supported by National Key R&D Program of China (2021YFA1000403), the National Natural Science Foundation of China (Nos. 11991022), the Strategic Priority Research Program of Chinese Academy of Sciences (Grant No. XDA27000000) and the Fundamental Research Funds for the Central Universities.

References

[1] Agarwal Y K, Venkateshan P. New valid inequalities for the optimal communication spanning tree problem. INFORMS Journal on Computing, 2019, 31: 268–284
[2] Ahmed R, Turja M A, Sahneh F D, et al. Computing Steiner trees using graph neural networks. arXiv:2108.08368, 2021
[3] Andrade R, Lucena A, Maculan N. Using Lagrangian dual information to generate degree constrained spanning trees. Discrete Applied Mathematics, 2006, 154: 703–717
[4] Arora S. Polynomial time approximation schemes for Euclidean traveling salesman and other geometric problems. Journal of the ACM, 1998, 45: 753–782
[5] Bastos M P, Ribeiro C C. Reactive tabu search with path-relinking for the Steiner problem in graphs. Boston, MA: Springer US, 2002, 15: 39–58
[6] Bello I, Pham H, Le Q V, et al. Neural combinatorial optimization with reinforcement learning. arXiv preprint arXiv:1611.09940, 2017
[7] Bicalho L H, da Cunha A S, Lucena A. Branch-and-cut-and-price algorithms for the degree constrained minimum spanning tree problem. Computational Optimization and Applications, 2016, 63: 755–792
[8] Boldon B, Kumar N. Minimum-weight degree-constrained spanning tree problem: heuristics and implementation on an SIMD parallel machine. Parallel Computing, 1996, 14
[9] Bonnet É, Sikora F, The PACE 2018 parameterized algorithms and computational experiments challenge: The third iteration. In: IPEC, 2018
[10] Borůvka O, O jisém problému minimálním, 1926., 24
[11] Bouchachia A, Prosesgeger M. A hybrid ensemble approach for the Steiner tree problem in large graphs: A geographical application. Applied Soft Computing, 2011, 11: 5745–5754
[12] Bresson X, Laurent T. The Transformer network for the traveling salesman problem. arXiv preprint arXiv:2103.03012, 2021
[13] Bui T N, Deng X H, Zrnčić C M. An improved ant-based algorithm for the degree-constrained minimum spanning tree problem. IEEE Transactions on Evolutionary Computation, 2012, 16: 266–278
[14] Bui T N, Zrnčić C M. An ant-based algorithm for finding degree-constrained minimum spanning tree. In: Proceedings of the 8th Annual Conference on Genetic and Evolutionary Computation - GECCO ’06, Seattle, Washington, USA: ACM Press, 2006, 11
[15] Byrka J, Grandoni F, Rothvoß T, et al. An improved LP-based approximation for steiner tree. In: Proceedings of the 42nd ACM Symposium on Theory of Computing - STOC ’10, Cambridge, Massachusetts, USA: ACM Press, 2010, 583
[16] Caccetta L, Hill S P. A branch and cut method for the degree-constrained minimum spanning tree problem. Networks, 2001, 37: 74–83
[17] Campos R, Ricardo M. A fast algorithm for computing minimum routing cost spanning trees. Computer Networks, 2008, 52: 3229–3247
[18] Chlebík M, Chlebíková J. The Steiner tree problem on graphs: Inapproximability results. Theoretical Computer Science, 2008, 406: 207–214
[19] Choque J N C. Optimal communication spanning tree. Mestrado em Ciência da Computação, Universidade de São Paulo, São Paulo, 2021
[20] da Cunha A S, Lucena A. Lower and upper bounds for the degree-constrained minimum spanning tree problem. Networks, 2007, 50: 55–66
[21] Dai H J, Khalil E B, Zhang Y Y, et al. Learning combinatorial optimization algorithms over graphs. arXiv:1704.01665, 2018
[22] de Aragão M P, Uchoa E, Werneck R F. Dual heuristics on the exact solution of large Steiner problems. Electronic Notes in Discrete Mathematics, 2001, 7: 150–153
[23] Delbem A C B, de Carvalho A, Policastro C A, et al. Node-depth encoding for evolutionary algorithms applied to network design. In: Kanade T, Kittler J, Kleinberg J M, Mattern F, Mitchell J C, Naor M, Nierstrasz O, Rangan C P, Steffen B, Sudan M, Terzopoulos D, Tygar D, Vardi M Y, Weikum G, Deb K, eds. Genetic and Evolutionary Computation – GECCO 2004, Berlin, Heidelberg: Springer Berlin Heidelberg, 2004, 3102: 678–687
[24] Ding M, Han C Y, Guo T D. High generalization performance structured self-attention model for knapsack problem. Discrete Mathematics, Algorithms and Applications, 2021, 13: 2150076
[25] Doan M N. An effective ant-based algorithm for the degree-constrained minimum spanning tree problem. In: 2007 IEEE Congress on Evolutionary Computation, Singapore: IEEE, 2007, 485–491
[26] Drori I, Kharkar A, Sickinger W R, et al. Learning to solve combinatorial optimization problems on real-world graphs in linear time. arXiv:2006.03750, 2020
[27] Du H Z, Yan Z, Xiang Q, et al. Vulcan: Solving the Steiner tree problem with graph neural networks and deep reinforcement learning. arXiv:2111.10810, 2021
[28] Duin C, Voß S. Efficient path and vertex exchange in steiner tree algorithms. Networks, 1997, 29: 89–105
[29] Esbensen H. Computing near-optimal solutions to the steiner problem in a graph using a genetic algorithm. Networks, 1995, 26: 173–185
[30] Fages J G, Lorca X, Rousseau L M. The salesman and the tree: the importance of search in CP. Constraints, 2016, 21: 145–162
[31] Fischetti M, Lancia G, Serafini P. Exact algorithms for minimum routing cost trees. Networks, 2002, 39: 161–173
[32] Furer M, Raghavachari B. Approximating the minimum-degree Steiner tree to within one of optimal. Journal of Algorithms, 1994, 17: 409–423
[33] Garey M R, Johnson D S. Computers and Intractability: A Guide to the Theory of NP-completeness, 27th ed., ser. A Series of Books in the Mathematical Sciences. New York: Freeman, 1979.
[34] Goemans M. Minimum bounded degree spanning trees. In 2006 47th Annual IEEE Symposium on Foundations of Computer Science (FOCS’06), Berkeley, CA, USA: IEEE, 2006, 273–282
[35] Guo T D, Han C Y, Tang S Q. Machine learning methods for combinatorial optimization. Beijing: Science Press, 2019
[36] Guo T D, Han C Y, Tang S Q, et al. Solving combinatorial problems with machine learning methods. In: Du D Z, Pardalos P M, Zhang Z, eds. Nonlinear Combinatorial Optimization, Cham: Springer International Publishing, 2019, 147: 207–229
[37] Hubbs C D, Perez H D, Sarwar O, et al. OR-Gym: A reinforcement learning library for operations research problems. arXiv preprint arXiv:2008.06319, 2020
[38] Huy N V, Nghia N D. Solving graphical Steiner tree problem using parallel genetic algorithm. In: 2008 IEEE International Conference on Research, Innovation and Vision for the Future in Computing and Communication Technologies. Ho Chi Minh City, Vietnam: IEEE, 2008, 29–35
[39] Johnson D S, Lenstra J K, Kan A H G R. The complexity of the network design problem. Networks, 1978, 8: 279–285
[40] Julstrom B A. The blob code is competitive with edge-sets in genetic algorithms for the minimum routing cost spanning tree problem. In: Proceedings of the 2005 Conference on Genetic and Evolutionary Computation - GECCO ’05, Washington DC, USA: ACM Press, 2005: 585-590
[41] Kapsalis A, Rayward-Smith V J, Smith G D. Solving the graphical Steiner tree problem using genetic algorithms. The Journal of the Operational Research Society, 1993 44: 397–406
[42] Karp R M. Reducibility among combinatorial problems. In: Complexity of computer computations, 1972, 85–103
[43] Khakhulin T, Schustki R, Oseledets I. Learning elimination ordering for tree decomposition problem. In: Learning Meets Combinatorial Algorithms at NeurIPS2020. 2020
[44] Kingma D P, Ba J. Adam: A method for stochastic optimization. arXiv preprint arXiv:1412.6980, 2017
[45] Koch T, Martin A. Solving Steiner tree problems in graphs to optimality. Networks, 1998, 32: 207–232
[46] Kool W, Van Hoof H, and Welling M. Attention, learn to solve routing problems!. arXiv:1803.08475, 2019
[47] Kruskal J B, On the shortest spanning subtree of a graph and the traveling salesman problem. 1956
[48] Kwon Y D, Choo J, Kim B, et al. POMO: Policy optimization with multiple optima for reinforcement learning. Advances in Neural Information Processing Systems, 2020, 33: 21188–21198
[49] Leitner M, Ljubić I, Luipersbeck M, et al. A dual ascent-based branch-and-bound framework for the prize-collecting Steiner tree and related problems. INFORMS Journal on Computing, 2018, 30: 402–420
[50] Lucena A, Beasley J E. A branch and cut algorithm for the Steiner problem in graphs. Networks, 1998, 31: 39–59
[51] Masone A, Nenni M E, Sforza A, et al. The minimum routing cost tree problem: State of the art and a core-node based heuristic algorithm. Soft Computing, 2019, 23: 2947–2957
[52] Narula S C, Ho C A, Degree-constrained minimum spanning tree. Computers & Operations Research, 1980, 7: 239–249
[53] Nazari M, Oroojlooy A, Snyder L V , et al. Reinforcement learning for solving the vehicle routing problem. Advances in Neural Information Processing Systems, 2018, 31
[54] Polzin T, Daneshgoz S V. Improved algorithms for the Steiner problem in networks. Discrete Applied Mathematics, 2001, 112: 263–300
[55] Presti G L, Re G L, Storniolo P, et al. A grid enabled parallel hybrid genetic algorithm for SPN. In: Kanade T, Kittler J, Kleinberg J M, Mattern F, Mitchell J C, Naor M, Nierstrasz O, Rangan C P, Steffen B, Sudan M, Terzopoulos D, Tygar D, Vardi M Y, Weikum G, Bubak M, van Albada G D, Sloop P M A, Dongarra J, eds. Computational Science - ICCS 2004, Berlin, Heidelberg: Springer Berlin Heidelberg, 2004, 3036: 156–163
[56] Prim R C. Shortest connection networks and some generalizations. Bell System Technical Journal, 1957, 36: 1389–1401
[57] Qu R, Xu Y, Castro J P, et al. Particle swarm optimization for the Steiner tree in graph and delay-constrained multicast routing problems. Journal of Heuristics, 2013, 19: 317–342
[58] Raidl G R, Julstrom B A. Edge sets: an effective evolutionary coding of spanning trees. IEEE Transactions on Evolutionary Computation, 2003, 7: 225–239
[59] Ribeiro C C, De Souza M C, Tabu search for the Steiner problem in graphs. Networks, 2000, 36: 138–146
[60] Ribeiro C C, Uchoa E, Werneck R F. A hybrid GRASP with perturbations for the Steiner problem in graphs. INFORMS Journal on Computing, 2002, 14: 228–246
[61] Sattari S, A metaheuristic algorithm for the minimum routing cost spanning tree problem. 2015, 14
[62] Sattari S, Didehvar F. Variable neighborhood search approach for the minimum routing cost spanning tree problem. Int J Oper Res, 2013, 10: 153–160
[63] Scott A J. The optimal network problem: some computational procedures. Transportation Research, 1969, 3: 201–210
[64] Singh A. A new heuristic for the minimum routing cost spanning tree problem. In: 2008 International Conference on Information Technology, Bhunaneswar, Orissa, India: IEEE, 2008, 9–13
[65] Singh A, Sundar S. An artificial bee colony algorithm for the minimum routing cost spanning tree problem. Soft Computing, 2011, 15: 2489–2499
[66] Singh M, Lau L C. Approximating minimum bounded degree spanning trees to within one of optimal. Journal of the ACM, 2015, 62: 1–19
[67] Singh K, Sundar S. A hybrid genetic algorithm for the degree-constrained minimum spanning tree problem. Soft Computing, 2020, 24: 2169–2186
[68] Sutton R S, McAllester D, Singh S, et al. Policy gradient methods for reinforcement learning with function approximation. Advances in Neural Information Processing Systems, 1999, 12
[69] Takahashi H, Matsuyama A. An approximate solution for the Steiner problem in graphs, 1980, 5
[70] Tan Q P. A genetic approach for solving minimum routing cost spanning tree problem. International Journal of Machine Learning and Computing, 2012, 410–414

[71] Tan Q P. A heuristic approach for solving minimum routing cost spanning tree problem. International Journal of Machine Learning and Computing, 2012, 406–409

[72] Tilk C, Irnich S, Combined column-and-row-generation for the optimal communication spanning tree problem. Computers & Operations Research, 2018, 93: 113–122

[73] Uchoa E, Werneck R F. Fast local search for the steiner problem in graphs. ACM Journal of Experimental Algorithmics, 2012, 17: 2–1

[74] Vaswani A, Shazeer N, Parmar N, et al. Attention is all you need. Advances in Neural Information Processing Systems, 2017, 30

[75] Vinyals O, Fortunato M, Jaitly N. Pointer networks. Advances in Neural Information Processing Systems, 2015, 28

[76] Wang C G, Han C Y, Guo T D, et al. Solving uncapacitated p-median problem with reinforcement learning assisted by graph attention networks. Applied Intelligence, 2022

[77] Wang C G, Yang Y D, Slumbers O, et al. A game-theoretic approach for improving generalization ability of TSP solvers. [arXiv:2110.15105] 2022

[78] Wang Q, Hao Y S, Cao J. Learning to traverse over graphs with a Monte Carlo tree search-based self-play framework. Engineering Applications of Artificial Intelligence, 2021, 105: 104422

[79] Wong R T. Worst-case analysis of network design problem heuristics. SIAM Journal on Algebraic Discrete Methods, 1980, 1: 51–63

[80] Wu B Y, Lancia G, Bafna V, et al. A polynomial-time approximation scheme for minimum routing cost spanning trees. SIAM Journal on Computing, 2000, 29: 761–778

[81] Zetina C A, Contreras I, Fernández E, et al. Solving the optimum communication spanning tree problem. European Journal of Operational Research, 2019, 273: 108–117

[82] Zheng W J, Wang D L, Song F G. OpenGraphGym: A parallel reinforcement learning framework for graph optimization problems. In: Krzhizhanovskaya V V, Závodszky G, Lees M H, Dongarra J J, Sloat P M A, Brissos S, Teixeira J, eds. Computational Science – ICCS 2020, Cham: Springer International Publishing, 2020, 12141: 439–452