NN-Steiner: A Mixed Neural-Algorithmic Approach for the Rectilinear Steiner Minimum Tree Problem

Andrew B. Kahng[1, 2], **Robert R. Nerem**[3], **Yusu Wang**[3], **and Chien-Yi Yang**[2]

[1]Department of Electrical and Computer Engineering, University of California San Diego
[2]Department of Computer Science and Engineering, University of California San Diego
[3]Halıcıoğlu Data Science Institute, University of California San Diego
abk@ucsd.edu, rrnerem@ucsd.edu, yusuwang@ucsd.edu, chy036@ucsd.edu

Abstract

Recent years have witnessed rapid advances in the use of neural networks to solve combinatorial optimization problems. Nevertheless, designing the "right" neural model that can effectively handle a given optimization problem can be challenging, and often there is no theoretical understanding or justification of the resulting neural model. In this paper, we focus on the rectilinear Steiner minimum tree (RSMT) problem, which is of critical importance in IC layout design and as a result has attracted numerous heuristic approaches in the VLSI literature. Our contributions are two-fold. On the methodology front, we propose NN-Steiner, which is a novel **mixed neural-algorithmic framework** for computing RSMTs that leverages the celebrated PTAS algorithmic framework of Arora to solve this problem (and other geometric optimization problems). Our NN-Steiner replaces key algorithmic components within Arora's PTAS by suitable neural components. In particular, NN-Steiner only needs **four** neural network (NN) components that are called repeatedly within an algorithmic framework. Crucially, each of the four NN components is only of **bounded size independent of input size**, and thus easy to train. Furthermore, as the NN component is learning a generic algorithmic step, once learned, the resulting mixed neural-algorithmic framework generalizes to much larger instances not seen in training. Our NN-Steiner, to our best knowledge, is the first neural architecture of bounded size that has capacity to approximately solve RSMT (and variants). On the empirical front, we show how NN-Steiner can be implemented and demonstrate the effectiveness of our resulting approach, especially in terms of generalization, by comparing with state-of-the-art methods (both neural and non-neural based).

1 Introduction

Given a set of points V in $\mathbb{R}^d$, a Steiner tree spanning V is a tree whose vertex set is V together with a set of additional points $S \subset \mathbb{R}^d$ called *Steiner points*. A *rectilinear Steiner tree* is a Steiner tree where all edges are axis-parallel. Given V, the *rectilinear Steiner minimum tree (RSMT)* problem aims to compute the rectilinear Steiner tree spanning V with smallest possible cost, defined as the total length of all edges in the tree. The RSMT problem has fundamental importance in VLSI physical design, as minimum wiring is correlated with key figures of merit including dynamic power, congestion, and timing delay. Hence, RSMT constructions have been well-studied for interconnect planning and estimation, timing estimation, global routing, and other applications (Kahng et al. 2022).

The existence of an optimal RSMT whose Steiner points are restricted to the *Hanan grid*, which is formed by intersections of all axis-parallel lines passing through points in V, was established in (Hanan 1966). The RSMT problem was subsequently shown to be NP-complete (Garey and Johnson 1977). It was proved by (Hwang 1976) that the rectilinear minimum spanning tree gives a $3/2$-approximation. Results leveraging Zelikovsky's method led to a $5/4$-approximation (Berman et al. 1994). Theoretically, the best known approximation algorithm for RSMT in fixed-dimensional space is obtained via the PTAS (polynomial-time approximation scheme) proposed by Arora (Arora 1998). Arora's method provides a $(1+\varepsilon)$-approximation for a range of problems, such as the traveling salesperson problem, in addition to RSMT. Unfortunately, while this algorithm runs in time polynomial in the number of points, its time complexity depends exponentially on $\frac{1}{\varepsilon}$ and, consequently, the method has yet to find use in practice.

On one hand, given the importance of the RSMT problem in chip design and its intractability, a large number of heuristics have been developed in the VLSI CAD community, aiming to improve the quality of RSMT computation with practical running time, e.g., (Kahng, Măndoiu, and Zelikovsky 2003; Liu, Chen, and Young 2021; Hu et al. 2006; Fallin et al. 2022; Wang et al. 2005; Cinel and Bazlamaçci 2008). The current state-of-the-art (SOTA) heuristic algorithm is FLUTE (Wong and Chu 2008; Chu and Wong 2007, 2005; Chu 2004). FLUTE constructs a lookup table encoding optimal RSMTs for all instances smaller than 10 and then constructs RSMTs for larger pointsets by partitioning the input pointset into subsets of size q or smaller, and combining the optimal trees over these subsets. FLUTE has been shown to be very close to optimal for small pointsets and is widely used in practice in VLSI CAD. GeoSteiner (Juhl et al. 2018) is the SOTA algorithm for exact RSMT computation, and is an integer linear programming approach.

On the other hand, with recent success of deep neural networks (NNs) in many applications, there has been a surge in use of NNs to help tackle combinatorial optimization problems (Bengio, Lodi, and Prouvost 2021; Khalil et al.

2017; Li, Chen, and Koltun 2018; Selsam et al. 2018; Gasse et al. 2019; Sato, Yamada, and Kashima 2019), such as traveling salesperson or other routing-related problems, using reinforcement learning (RL) (Vinyals, Fortunato, and Jaitly 2015; Bello et al. 2017; Deudon et al. 2018; Prates et al. 2019). Recently, REST (Liu, Chen, and Young 2021) achieved the first NN-based approach for RSMT by finding so-called rectilinear edge sequences using RL. (Chen et al. 2022) designed an RL framework to find obstacle-avoiding Steiner minimum trees. Significant challenges in neural combinatorial optimization (NCO) remain. NNs are often used in an ad-hoc manner with limited theoretical understanding of the resulting framework. It is also often not known if machine-learning pipelines have the capacity to solve a given combinatorial optimization problem, or how network-architecture design could leverage problem structure to design more effective and efficient neural models.

One potential way to inject theoretical justification into the design of neural approaches for combinatorial problems is to leverage the vast literature on approximation algorithms. In particular, instead of using one NN to solve an optimization problem in an end-to-end manner, one can use neural components in a high-level algorithmic framework. An exemplary thread of works uses NNs to learn variable-selection decisions in branch-and-bound frameworks solving mixed-integer linear programming problems (Gasse et al. 2019; Gupta et al. 2020; Nair et al. 2020). (McCarty et al. 2021) propose a mixed neural-algorithmic framework *NN-Baker* to solve problems such as maximum independent set in the geometric setting by using Baker's technique to decompose problems into small instances of *bounded size*, and then training a single NN to solve these instances.

This Work. In this paper, we develop an approach to compute RSMTs in $\mathbb{R}^d$. (While we use $\mathbb{R}^2$ and Manhattan geometry, the framework extends to $\mathbb{R}^d$ for any constant value of d.) Specifically, we develop NN-Steiner[1], a mixed neural-algorithmic framework that leverages the ideas behind Arora's PTAS for RSMT (Arora 1998).

At a high level, Arora's PTAS partitions the input domain in a hierarchical manner, then solves the problem via a bottom-up dynamic programming (DP) procedure. One key result of (Arora 1998) is that each DP subproblem is of only bounded size, and thus, can be solved in time independent of the number of input points. However, the complexity of this DP step is prohibitive in practice.

In Section 3, We develop a mixed neural-algorithmic approach to simulate Arora's PTAS; Fig. 4 gives a high-level illustration. The costly DP step is replaced by a single NN component that outputs a learned embedding of the solutions to the DP subproblems. This **single NN module** is called repeatedly within the algorithmic framework. Other NN components simulate the backward retrieval of Steiner points in a top-down manner. As each of the four NN components is of size independent of input problem size, the model complexity of each component is bounded.

[1]Code is open-sourced at https://github.com/ABKGroup/NN-Steiner.

On the theoretical front, we show in Theorem 3.1 that this framework has the capacity to produce approximate RSMTs using only NNs of bounded complexity. On the practical front, NNs replace a key but costly component in Arora's PTAS, leading to an efficient architecture. Furthermore, since the neural component only needs to handle fixed-size instances, training is easier. Once trained, NN-Steiner generalizes well to problems of much larger sizes than seen in training, as we demonstrate in Section 4. Indeed, extensive experimental results show that NN-Steiner achieves better performance than NN-based and non-NN-based SOTA methods for sufficiently large problem sizes. NN-Steiner outperforms the existing RL-based neural approach significantly for large pointsets. As input size increases, the RL-based policy must handle a larger action space, which makes it challenging both to learn the policy and to generalize. NN-Steiner learns an algorithmic component of fixed size, leading to superior generalization.

In summary, we propose NN-Steiner, a novel neural-algorithmic framework for the RSMT problem, which leverages the algorithmic idea of Arora's PTAS (which is the theoretical best approximation algorithm for this problem). NN-Steiner is, to our best knowledge, the first neural architecture of bounded size that has capacity to approximately solve the RSMT problem. Moreover, the algorithmic alignment of NN-Steiner leads to better practical performance than existing SOTA methods for large instances. While we focus on the RSMT problem in this paper due to its practical importance in VLSI, the versatility of Arora's framework means our methodology can be potentially applied to other geometric optimization problems, such as the obstacle-avoiding RSMT problem (Ajwani, Chu, and Mak 2010).

Our work is one of the first NCO frameworks to use algorithmic alignment to remove dependence on problem size. To our best knowledge, the only other work in this direction is NN-Baker (McCarty et al. 2021), which is limited to a very simple algorithmic setup (a flat partitioning of the input domain). The dynamic programming (DP) framework we consider is much more general: for example, a similar DP framework exists for many optimization problems (e.g., max independent set) for graphs with bounded tree width (Cygan et al. 2015).

Removing problem-size dependence is important, as size generalization is a fundamental obstacle in NCO (Garmendia, Ceberio, and Mendiburu 2022) that is challenging to overcome (Liu et al. 2022). Training on large instances is prohibitively expensive: for supervised learning this requires computation of exact solutions to large instances, and for RL and unsupervised learning, training becomes exponentially more challenging as size increases. Thus, size generalization is essential for performance on large instances. In some cases, such generalization is even provably hard (Yehudai et al. 2020). Notably, our experiments show NN-Steiner exhibits strong size generalization on a hard optimization problem that has practical implications.

2 Preliminaries

We now introduce the RSMT problem, then briefly describe Arora's PTAS for this problem (Arora 1998). For simplicity,

we henceforth treat the case where input points lie in $\mathbb{R}^2$. Our definitions and Arora's algorithm can both be extended to $\mathbb{R}^d$, as well as to the standard Euclidean Steiner minimum tree problem (without rectilinear constraints).

Definition 1 (RSMT). *Given a set of points $V \subset \mathbb{R}^2$, the rectilinear Steiner minimum tree (RSMT) for V is a tree T with vertex set $V \cup S \subset \mathbb{R}^2$ with minimum total edge length under the ℓ_1 norm. The set S is the set of Steiner points.*

Figure 1: (L) Rectilinear spanning tree of input (blue) points. (R) Rectilinear Steiner minimum tree (red points are Steiner points).

Arora's PTAS

We now describe the high-level idea behind Arora's polynomial-time approximation algorithm, which we refer to as *Arora's PTAS*. For simplicity, we assume that the input points $V \subset \mathbb{R}^2$ have integral coordinates, and are contained in a bounding box of side length $L = O(n)$ with $n = |V|$. A perturbation process given in (Arora 1998) rounds input coordinates so that this assumption holds without changing theoretical guarantees for the algorithm.

Step 1: Construct a shifted quadtree. First, we pick integers $a, b \in [0, L)$ uniformly at random and then translate the input pointset by the vector (a, b). We then construct a quadtree, where the splitting of quadtree cells terminates if a cell contains 1 or 0 points. The quadtree is a tree Q where each internal vertex has degree 4. The root has level 0, and is associated with a cell of side length L. Any vertex $v \in Q$ at level i is associated with a cell A_v of size (i.e., side length) $\frac{L}{2^i}$. Bisecting a level-i cell A_v with a horizontal line and with a vertical line decomposes it into four level-$(i+1)$ child cells, each of size $\frac{L}{2^{i+1}}$, corresponding to the four children of $v \in Q$. As the side length of the bounding box is $L = O(n)$, and points have integral coordinates, the height (max-level) of the quadtree is at most $O(\log L)$ and the total number of vertices in Q (and thus the number of cells across all levels) is $O(n \log L) \subseteq O(n \log n)$.

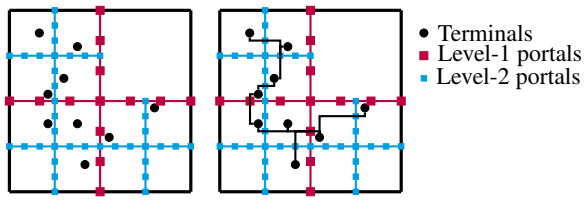

Figure 2: (a) A two-level quadtree over the input points (black dots), where each side of a quadtree cell has 4 portals. (b) An example of a $(2, 1)$-light rectilinear Steiner tree.

We now consider a special family of Steiner trees for which the crossing of quadtree cells is constrained.

Definition 2. *Let m, r be positive integers. The m-regular set of portals is the set of points such that each cell has a portal at each of its 4 corners, and m other equally-spaced portals on each of its four sides. A Steiner tree is (m, r)-light if it crosses each edge of each cell at most r times and always at a portal.*

Note that if a side of a cell S is contained in the sides of multiple cells, then the portals on S are spaced according to the cell with side containing S that has the lowest level i. We say that the portals on S have level i. See Fig. 2 for an example of quadtree decomposition and a $(2, 1)$-light rectilinear Steiner tree.

Step 2: Dynamic programming. The following theorem guarantees that the minimal (m, r)-light rectilinear Steiner tree is an approximate RSMT (Arora 1998). A proof is given in the supplemental (Kahng et al. 2023).

Theorem 2.1 (Structure Theorem). *If shifts $0 \le a, b < L$ are chosen uniformly randomly in $[0, L)$, then with probability at least $1/2$, the minimum-length (m, r)-light rectilinear Steiner tree has length at most $\left(1 + \frac{4}{r} + O(\frac{4 \log L}{m})\right)\mathsf{OPT}$, where OPT is the length an RSMT.*

Hence, our goal is to compute a minimum (m, r)-light rectilinear Steiner tree. In particular, Arora proposed to use DP in a bottom-up construction. We sketch the idea here.

We process all quadtree cells in a bottom-up manner. For a fixed quadtree cell A, consider an (m, r)-light Steiner tree T restricted to A; this gives rise to some Steiner forest T_{A} in A, which can exit this cell only via portals on sides of A. In particular, the portion of the Steiner tree outside A can be solved independently as long as we know the following *portal configuration*: (i) the set of *exiting portals* on the side of this cell that are used by T (which connect points outside A with those inside), and (ii) how these exiting portals are connected by trees in the Steiner forest T_{A}.

Let Ξ_{A} be the set of portal configurations for a cell A, and let $D = |\Xi_{\mathsf{A}}|$. Each cell has $(4m+4)^{4r}$ subsets of $4r$ portals, and each subset of portals can be partitioned $\mathsf{Bell}(4r)$ ways. As the Bell number $\mathsf{Bell}(k)$ is bounded above by k^k we have $D = (4m+4)^{4r}\mathsf{Bell}(4r) < (4m+4)^{8r}$. Our goal is to compute, for each portal configuration $\sigma \in \Xi$, the minimum cost $\mathsf{cost}(\sigma)$ of any rectilinear Steiner forest within A that gives rise to this boundary condition. Assuming an arbitrary but fixed order of portal configurations in $\Xi = \{\sigma_1, \ldots, \sigma_D\}$, the costs of all configurations can then be represented by a vector $\vec{\mathsf{C}}_{\mathsf{A}} \in \mathbb{R}^D$, where $\vec{\mathsf{C}}_{\mathsf{A}}[i] = \mathsf{cost}(\sigma_i)$. We call $\vec{\mathsf{C}}_{\mathsf{A}}$ the *cost-vector for* A.

We now describe the DP algorithm to compute this cost-vector for all cells in a bottom-up manner (decreasing order of levels).

Base case: A *is a leaf cell.* In this case, there is at most one point p from V contained in A. We enumerate all configurations, and compute $\vec{\mathsf{C}}_{\mathsf{A}}$ directly, which requires solving RSMT instances of bounded size.

Inductive step: A *is not a leaf cell.* The four child-cells $\mathsf{A}_1, \ldots, \mathsf{A}_4$ of A are from the level below A's level, and thus, by the inductive hypothesis, we have already computed the

cost-vectors $\vec{\mathsf{C}}_{\mathsf{A}_i}$ for $i = 1, \ldots, 4$. Consider any portal configuration $\sigma \in \Xi$. We simply need to enumerate all choices of portal configurations $\tau_1, \ldots, \tau_4$ for child-cells $\mathsf{A}_1, \ldots, \mathsf{A}_4$ that are *consistent with* σ, meaning that the portals on common sides are the same, and the connected components of portals from the 4 child-cells do not form cycles (hence, still induce a valid Steiner forest). We have

$$\mathsf{cost}(\sigma) = \min_{\tau_1,\ldots,\tau_4 \text{ consistent with } \sigma} \sum_{i\in\{1,2,3,4\}} \mathsf{cost}[\tau_i] \qquad (1)$$

Final construction of approximate RSMT. At the end of DP, after we compute the cost-vector for the root cell, we identify the portal configuration $\sigma^* \in \Xi$ with lowest cost. To obtain the corresponding rectilinear Steiner tree, we perform a top-down backtracking: (i) for the root cell, from σ^* we can retrieve the set of child-cell configurations $\tau_1^*, \ldots, \tau_4^*$ generating σ^*; (ii) we repeat this until we reach all leaf cells. At each leaf cell we once again compute the optimal Steiner forest for the chosen configuration. Combining these Steiner forests yields an optimal (m, r)-light RSMT.

3 NN-Steiner

Although the running time of Arora's PTAS is polynomial in the problem size, its computation is prohibitively expensive in practice as we compute all possible portal configurations. Instead of brute-force enumeration of portal configurations, we propose a framework, *NN-Steiner*, which infuses neural networks (NNs) into Arora's PTAS to select Steiner points from portals to build the output Steiner tree. First, we show that the key components within the DP algorithm can be simulated exactly by certain NNs. However, such NNs are not efficient either, as their size depends exponentially on m and r. We then show a practical instantiation of NN-Steiner and demonstrate its performance in Sec. 4.

Theoretical NN-Steiner to Simulate Arora's PTAS

We can simulate key components in the DP framework of Arora's PTAS. Specifically, we use four neural networks: $\mathsf{NN}_{\mathsf{base}}$ and $\mathsf{NN}_{\mathsf{DP}}$ respectively implement the base case and inductive step of DP to compute encodings of cost-vectors in a bottom-up manner, while $\mathsf{NN}_{\mathsf{top}}$ and $\mathsf{NN}_{\mathsf{retrieve}}$ obtain optimal portal configurations from cells in a top-down manner.

There exist designs and parameters of these NNs simulating Arora's DP algorithm exactly; moreover, these NNs are each of only **bounded size** independent of n (depending only on parameters m and r, which are set to be constant in practice). Here, we describe how to construct $\mathsf{NN}_{\mathsf{DP}}$ to simulate one inductive step in the DP algorithm. For brevity, we leave the other NN cases to a full version.

Recall that, as described in Sec. 2, the inductive step of the DP algorithm can be rewritten as applying a function $f_{DP} : (\mathbb{R}^D)^4 \to \mathbb{R}^D$, where D is the total number of portal configurations for a single cell. In particular, for any quadtree cell A with child-cells $\mathsf{A}_1, \ldots, \mathsf{A}_4$, the input to f_{DP} is the four cost-vectors $(\vec{\mathsf{C}}_{\mathsf{A}_1}, \vec{\mathsf{C}}_{\mathsf{A}_2}, \vec{\mathsf{C}}_{\mathsf{A}_3}, \vec{\mathsf{C}}_{\mathsf{A}_4}) \in (\mathbb{R}^D)^4$, and the output is the cost-vector $\vec{\mathsf{C}}_{\mathsf{A}} \in \mathbb{R}^D$.

Eqn. (1) gives how to compute each entry in the output vector $f_{DP}(\vec{\mathsf{C}}_{\mathsf{A}_1}, \ldots, \vec{\mathsf{C}}_{\mathsf{A}_4})$. That is, f_{DP} has a simple form modeled as a certain linear function followed by a min-pooling, which can be simulated by a NN as shown in Fig. 3. In particular, to compute $\vec{\mathsf{C}}_{\mathsf{A}}[\sigma_i]$, each neuron c_ℓ takes in a set of four portal configurations $\tau_j \in \mathsf{A}_j$, $j = 1, \ldots, 4$, consistent with σ_i, and simply does a sum operation. Then the output takes a min-pooling over all values at c_ℓs. As there are at most D^4 sets of four portal configurations for each σ_i, the entire model has complexity $\Theta(D^5)$. (Recall that $D \le (4m+4)^{8r}$ is independent of the input pointset size n.) That is, there is a $\mathsf{NN}_{\mathsf{DP}}$ of bounded complexity simulating the DP step exactly. The following theorem summarizes the existence of NNs to implement Arora's PTAS.

Theorem 3.1. *There exist four NNs, each of only bounded size depending only on m and r, that can simulate the DP algorithm of Arora's PTAS, such that the resulting mixed neural-algorithmic framework can find a $\big(1 + \frac{4}{r} + O(\frac{4\log L}{m})\big)$-approximate rectilinear Steiner tree (i.e., with length at most $\big(1 + \frac{4}{r} + O(\frac{4\log L}{m})\big)\mathsf{OPT}$). The framework calls these NNs only $O(n \log L) \subseteq O(n \log n)$ times.*

Practical Instantiation of NN-Steiner

The theoretical neural-algorithmic framework described in the previous section is not practical as it is explicitly encoding the exponential number of portal configurations (exponential in m, r). We now present NN-Steiner, a practical instantiation of this framework. Theorem 3.1 implies that the NN-Steiner architecture has the capacity to approximately solve the RSMT problem using NN components of bounded size. In practice, one hopes that NN-Steiner can leverage data to encode portal configurations more efficiently.

The pipeline starts with a neural network $\mathsf{NN}_{\mathsf{base}}$, that acts on each leaf cell to produce an encoding of the configuration costs. Next, we apply a NN simulation of the dynamic programming step $\mathsf{NN}_{\mathsf{DP}}$, for each non-leaf cell. We then use two NNs, $\mathsf{NN}_{\mathsf{top}}$ and $\mathsf{NN}_{\mathsf{retrieve}}$, to simulate the backtracking stage and return the likelihood that each portal is a Steiner point. By selecting high-likelihood portals, we construct a set of Steiner points and a corresponding Steiner tree. As the selected Steiner points S must lie on cell boundaries, we finish with a local refinement scheme which introduces Steiner points that lie in the interior of leaf cells. See Fig. 4 for a high-level overview of our practical NN-Steiner pipeline.

Forward pass. The forward processing involves two MLPs, $\mathsf{NN}_{\mathsf{base}}$ and $\mathsf{NN}_{\mathsf{DP}}$, and calls $\mathsf{NN}_{\mathsf{DP}}$ recursively in a bottom-up manner to compute an *implicit encoding of the costs of possible portal configurations*.

Base-case neural network $\mathsf{NN}_{\mathsf{base}}$. At the leaf level of Arora's PTAS, each cell contains at most 1 point. We instead terminate the quadtree decomposition when a cell contains no more than k_b points, where k_b is a hyperparameter. Given a leaf cell B with $V_{\mathsf{B}} \subset V$ the set of points contained in B, $\mathsf{NN}_{\mathsf{base}}$ takes relative coordinates of V_{B} as well as the set of portals on the sides of B as input, and outputs a d_c-dimensional vector as an implicit encoding of the cost vector $\vec{\mathsf{C}}_{\mathsf{B}} \in \mathbb{R}^D$; note that $\mathsf{d}_c << D$ in practice. Coordinates are specified relative to the bottom left corner of the cell, and are normalized by the cell size. If a leaf cell has $n_p < \mathsf{k}_b$

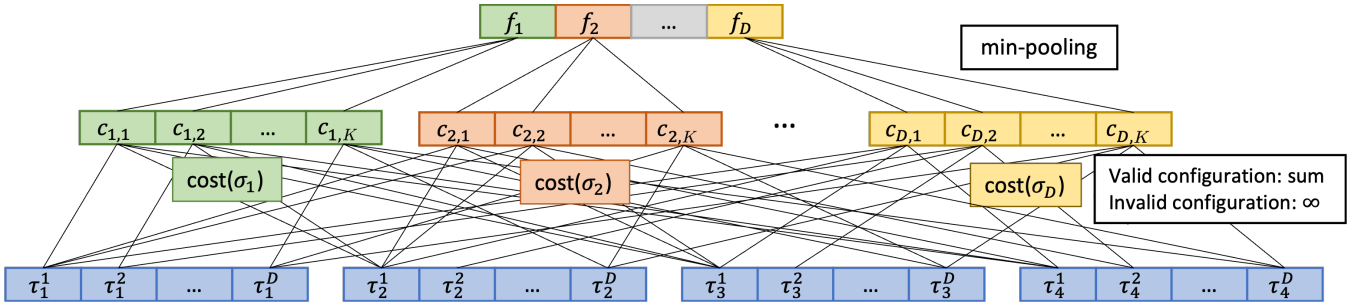

Figure 3: An NN simulating f_{DP}. Here, $K = D^4$ and $\vec{\mathsf{C}}_{\mathsf{A}_i}[\sigma_j] = \tau_i^j$ encodes the j^{th} configuration cost in cell A_i. We sum over costs A_i $(i = 1, 2, 3, 4)$ consistent with a portal configuration σ_j $(j = 1, 2, ..., D)$ so that the resulting vector cost(σ_j) encodes the cost of all configurations consistent with σ_j. Min-pooling on cost(σ_j) yields the vector $\vec{\mathsf{C}}_{\mathsf{A}_i}$ for $i \in \{1, 2, 3, 4\}$ (blue).

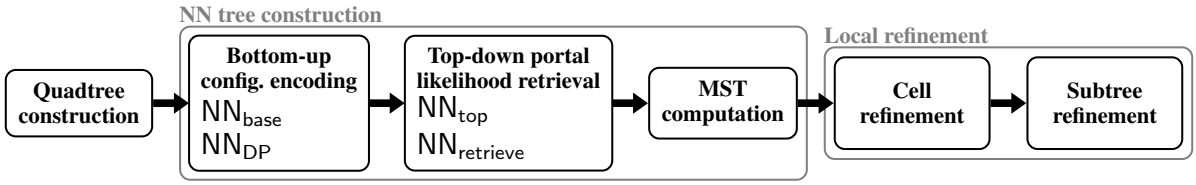

Figure 4: Pipeline of an NN-Steiner instantiation.

input points, we pad the remaining $\mathsf{k}_b - n_p$ coordinates with $(-1, -1)$. Each cell can have at most $4m + 8$ portals ($m + 2$ on each side) and these portals can appear only at $4m+8$ distinct relative locations in the cell. Portals are then provided to $\mathsf{NN}_{\mathsf{base}}$ as $4m + 8$ indicators in $\{0, 1\}$. Here, each corner contains two portals to simplify computation by distinguishing connections passing through from different sides. (Arora's PTAS uses a single portal at each corner.)

Terminating with at most k_b points in each leaf cell is advantageous as, for a cell with few points, it is challenging to learn a meaningful encoding of portal configurations. If k_b is small, the majority of cells have few points. (Note that a complete degree-4 tree has around 75% of its nodes at the leaf level.) Hence, training on such a collection of cells provides bad supervision, harming the effectiveness of learning. On the other hand, as k_b increases, larger Steiner-tree instances must be computed at leaves, which may negatively affect the performance of the framework. In our experiments, k_b is a hyperparameter; see Sec. 4 for its analysis.

DP-inductive-step neural network $\mathsf{NN}_{\mathsf{DP}}$. Next, we use another MLP, $\mathsf{NN}_{\mathsf{DP}}$, to simulate the function f_{DP}, which is equivalent to the DP in Arora's PTAS. In detail, given a cell A at level i, let $\mathsf{A}_1, \ldots, \mathsf{A}_4$ be its 4 child-cells at level $i+1$ in a fixed order. The neural network $\mathsf{NN}_{\mathsf{DP}}$ takes encodings of $\vec{\mathsf{C}}_{\mathsf{A}_1}, \ldots, \vec{\mathsf{C}}_{\mathsf{A}_4}$, and the portals of A, and generates an encoding of the cost vector $\vec{\mathsf{C}}_{\mathsf{A}} \in \mathbb{R}^D$ of the parent cell A. Note that the encodings of $\vec{\mathsf{C}}_{\mathsf{A}_1}, \ldots, \vec{\mathsf{C}}_{\mathsf{A}_4}$ are produced by applying either $\mathsf{NN}_{\mathsf{DP}}$ or $\mathsf{NN}_{\mathsf{base}}$ at $\mathsf{A}_1, \ldots, \mathsf{A}_4$. Again, portals are provided via $4m + 8$ indicators in $\{0, 1\}$.

Backward pass. The forward pass applies $\mathsf{NN}_{\mathsf{base}}$ to all leaf cells and $\mathsf{NN}_{\mathsf{DP}}$ to all internal vertices, simulating the bottom-up stage of Arora's PTAS. Now, we simulate the backtracking stage of Arora's PTAS with two additional NNs. These NNs construct a *portal-likelihood map* $\rho : \mathcal{P} \to [0, 1]$, where $\mathcal{P}$ is the set of all portals.

Root-level retrieval neural network $\mathsf{NN}_{\mathsf{top}}$. For a cell A let $\mathrm{Portals}(\mathsf{A})$ denote the portals in A that are one level higher than A's level, i.e., the portals on the vertical and horizontal segments bisecting A. Let R be the root cell with children $\mathsf{A}_1, \ldots \mathsf{A}_4$. The input to $\mathsf{NN}_{\mathsf{top}}$ is the output of $\mathsf{NN}_{\mathsf{DP}}$ at R, and the input of $\mathsf{NN}_{\mathsf{DP}}$ at R. The output of $\mathsf{NN}_{\mathsf{top}}$ is a vector of the likelihoods of $\mathrm{Portals}(\mathsf{R})$, where R is the root cell.

Backward retrieval step neural network $\mathsf{NN}_{\mathsf{retrieve}}$. We compute the rest of the portal likelihoods in a top-down manner. This is achieved by using a neural network $\mathsf{NN}_{\mathsf{retrieve}}$ (at all non-root non-leaf cells A) which computes the likelihoods of $\mathrm{Portals}(\mathsf{A})$. The input of $\mathsf{NN}_{\mathsf{retrieve}}$ applied at a cell A with child-cells $\mathsf{A}_1, \ldots$ and A_4, is comprised of two parts. First, $\mathsf{NN}_{\mathsf{retrieve}}$ takes the output of four instances of either $\mathsf{NN}_{\mathsf{base}}$ or $\mathsf{NN}_{\mathsf{DP}}$ applied at $\mathsf{A}_1, \ldots$, and A_4 (same as $\mathsf{NN}_{\mathsf{top}}$). Second, $\mathsf{NN}_{\mathsf{retrieve}}$ receives likelihoods for every portal on the boundary of A. At this step, these likelihoods will have been computed previously by an instance of either $\mathsf{NN}_{\mathsf{retrieve}}$ or $\mathsf{NN}_{\mathsf{top}}$ applied at the level above A's level.

Retrieval of Steiner points and postprocessing. After the backward pass, we have a portal-likelihood map $\rho : \mathcal{P} \to [0, 1]$ over all portals. We select all portals with likelihood greater than a threshold $t \in (0, 1)$ as the initial set of Steiner points S. We then compute the minimum spanning tree over $S \cup V$, which takes time $O(|S \cup V| \log |S \cup V|)$. Next, we apply three local refinement steps:

1. First, we iterate over all leaf cells and for each cell replace every connected component with the optimal Steiner tree connecting all of the component's Steiner points (selected portals) and input points.

2. Next, we remove Steiner points with degree less than 3 and round the locations of the remaining Steiner points to integer coordinates.
3. Finally, we partition[2] the tree into subtrees with k or fewer input points. We replace each subtree with the optimal Steiner tree over both its input points and any Steiner point which is adjacent to a vertex in another partition.

Optimal Steiner trees are computed using GeoSteiner 5.1 (Juhl et al. 2018). The first step, cell refinement, introduces Steiner points into the interior of cells since initially Steiner points can only be at portals. The second step removes unnecessary Steiner points and rounds the locations of Steiner points. We perform this rounding step as the optimal Steiner points are known to lie on the Hanan grid and thus have Steiner points with integer coordinates. The final step, subtree refinement, allows Steiner points at portals to be moved.

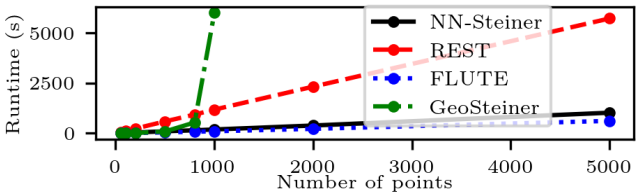

Figure 5: Total runtime for solving 100 pointsets.

Training. For each training pointset, we compute the optimal Steiner tree using GeoSteiner 5.1 (Juhl et al. 2018). Each time this optimal tree crosses a side of a cell, we move the crossing to the nearest portal. The resultant set of portals with crossings is used as the target in computing the loss. For the loss, we use binary cross entropy with weights of $m + 1$ for portals that are Steiner points in the target, and weights of 1 for other portals. As the classification is imbalanced, with most portals not being Steiner points, this scheme prevents low loss with uniformly near-zero portal likelihoods.

We train all four networks in an end-to-end manner. In training, the models are connected in a tree structure similar to that used in RNNs (recursive neural networks) in that the output of $\mathsf{NN}_{\mathsf{DP}}$ is fed into the same $\mathsf{NN}_{\mathsf{DP}}$ instance repeatedly in our architecture. However, RNNs are often connected linearly while NN-Steiner connects NNs in a tree structure. To accelerate training, we utilize batch-mode training. Batch mode necessitates that the same tree structure is used for each training sample. Otherwise, the model shape would be different between training samples, and these samples could not be learned in parallel.

We use pointsets sampled from a distribution tailored for compatibility with batch-mode training. Pointsets are constructed as follows: (i) Sample n_0 points from a distribution $\mathcal{D}$ on an integer grid of size $N \times N$. (ii) Construct a depth-d quadtree. (iii) In each quadtree leaf cell with greater than k_b points, remove points randomly until only k_b points remain. With these pointsets, a depth-d quadtree can always be used with no more than k_b points in every leaf cell. Note that while we use a fixed tree structure throughout training, for testing, the tree structure is decided by the specific test pointset and can have any shape or depth.

[2]We partition by iterating the following: (i) select a leaf, (ii) use breadth-first search to select vertices until k input points or all remaining input points are selected, then (iii) remove these vertices. The subtree induced by each removed set of vertices, which contains at most k input points, forms an element of the partition.

4 Experimental Performance

We present experimental results of NN-Steiner on planar RSMTs. Our experiments show that NN-Steiner outperforms SOTA algorithms for large point sets and has approximately linear runtime. We demonstrate the effectiveness of NN-Steiner on differently distributed pointsets. We give ablations that elucidate the role of our portal-retrieval and refinement schemes and we show the dependence of NN-Steiner on critical hyperparameters. All of our experiments run on a 64-bit Linux server with a 2.25GHz AMD EPYC 7742 Processor (256 threads) and three Nvidia RTX A100-SXM4 GPUs allocating 80GB RAM.

Implementation. Each of $\mathsf{NN}_{\mathsf{base}}, \mathsf{NN}_{\mathsf{DP}}, \mathsf{NN}_{\mathsf{top}}, \mathsf{NN}_{\mathsf{retrieve}}$ is implemented using a 4-layer MLP, with ReLU activation and size-4096 hidden layers. The output of $\mathsf{NN}_{\mathsf{base}}$ and $\mathsf{NN}_{\mathsf{DP}}$ is a vector of dimension $\mathsf{d}_c = 16(4m + 8)$. An additional sigmoid layer is appended to $\mathsf{NN}_{\mathsf{top}}$ and $\mathsf{NN}_{\mathsf{retrieve}}$ to output the portal likelihoods in $[0, 1]$. For training, we use a dropout of 0.1 and a batch size of 5000. Training pointsets are generated using a uniform distribution with $n_0 = 180$ initial points, $N = 100$ grid lines, and using a quadtree of depth $d = 3$. (In testing, the quadtree may have depth much greater than d, depending on the number and distribution of points.) We train the models on 120,000 pointsets solved exactly by GeoSteiner. For the refinement step, we partition into subtrees of size $k = 10$. We train for 5000 epochs using Adam (Kingma and Ba 2014) as our optimizer, with a learning rate of 10^{-4}. The default setting of NN-Steiner is $m = 15$ and $k_b = 4$ with the threshold set to $t = .95$.

Performance comparison. The results of the exact RSMT solver GeoSteiner are ground truth solutions. Since we could not compute GeoSteiner for large instances, we use FLUTE (Chu and Wong 2007), the SOTA heuristic solver, as the base for comparison of approaches. Table 1 shows the performance comparison. Smaller values indicate better performance and negative values indicate superior performance to FLUTE, i.e., length is smaller than the output of FLUTE. All the FLUTE instances are run with $A = 18$, the setting which produces the highest-quality solutions. We also make comparisons against REST (Liu, Chen, and Young 2021) with $T = 8$, the best-performing setting claimed by the work. Batch sizes are set to 1. Test pointsets are generated uniformly at random from a $10^4 \times 10^4$ grid. Reported values are averages over 100 pointsets. Table 1 shows that NN-Steiner generalizes to large pointsets. (Recall that training pointsets have less than $n_0 = 180$ points.) Furthermore, NN-Steiner outperforms FLUTE and REST for pointsets of size 500 or greater, with margin of outperformance increasing with pointset size. NN-Steiner has advantage over FLUTE for large problems, however, it performs best on small point sets relative to the exact solution. Also, Fig. 5 shows that the runtime of NN-Steiner scales approximately linearly.

Number of points	50	100	200	500	800	1000	2000	5000
NN-Steiner	2.10	1.38	0.74	**-0.67**	**-1.11**	**-1.43**	**-2.44**	**-2.99**
REST (T=8)	**-0.14**	1.08	7.40	22.68	35.16	42.52	75.12	147.48
FLUTE (A=18)	0.00	**0.00**	**0.00**	0.00	0.00	0.00	0.00	0.00
GeoSteiner	-0.54	-1.23	-2.25	-3.71	-4.43	-4.78	–	–

Table 1: Performance comparison on uniformly distributed pointsets (average percent length difference compared to FLUTE).

Number of points	50	100	200	500	800	1000	2000	5000
NN tree construction + refinement (NN-Steiner)	2.10	1.38	0.74	-0.67	-1.11	-1.43	-2.44	-2.99
MST of V + refinement	2.54	2.50	1.44	0.04	-0.75	-1.04	-1.73	-2.46
NN tree construction + cell refinement	3.01	2.54	1.84	0.39	-0.10	-0.45	-1.39	-2.08
NN tree construction + subtree refinement	6.76	5.55	4.90	3.39	3.20	3.07	1.64	1.44
FLUTE + refinement	-0.03	-0.08	-0.15	-0.22	-0.27	-0.27	-0.31	-0.33

Table 2: Ablation experiments showing the effect of refinement (average percent length difference compared to FLUTE).

Generalization to Different Distributions. We test the generalization of NN-Steiner to different pointset distributions. In particular, NN-Steiner is trained on uniformly distributed pointsets, but tested on pointsets with mixed normal and non-isotropic normal distributions. Points are restricted to a $10^4 \times 10^4$ grid. The standard deviation of the non-isotropic normal distribution is taken to be 3000 for both x and y and 1500 for mixed-normal distribution. Means are distributed uniformly. The covariance matrix of the non-isotropic normal distribution is given by uniformly picking a correlation in $[-1, 1]$. For the mixed-normal distribution, we use a uniform mixture of 10 normal distributions.[3] Fig. 6 shows that NN-Steiner generalizes to different distributions, despite only being trained on uniformly distributed pointsets.

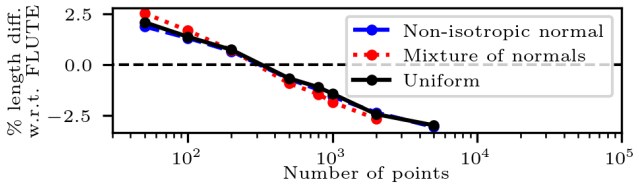

Figure 6: NN-Steiner performance on different distributions.

Ablations. We use an ablation study to show the importance of each component of our algorithm. To determine the impact of different components on performance as a whole, we evaluate the performance after removing each component. The components we consider are tree construction, cell refinement, and subtree refinement (Fig. 4). Tree construction produces an MST over $V \cup S$ where S is found by portal retrieval. To evaluate performance without this component, we instead apply refinement to the MST over V (here, edges between cells are preserved in cell refinement). Table 2 shows that each of these stages contributes to the overall performance, with refinement contributing the most. However, this observation does not undermine the importance of tree construction; if we apply refinement to the output of FLUTE (Chu and Wong 2007), also shown in Table 2, we see limited improvement. This suggests that the global topology produced by tree construction and cell refinement for large pointsets is superior to that produced by FLUTE.

Hyperparameter choice. We evaluate how m and k_b affect performance (Fig. 7). Results show poor performance for small values of k_b, which we attribute to overfitting. In theory, (m, r)-light trees are better approximations with greater values of m. However, for smaller instances, $m = 7$ yields similar performance as $m = 15$. This suggests that, for smaller instances, the fine granularity from larger m is not needed to generate high-quality global tree topologies. Also, models with smaller m may exhibit better performance as they are easier to train due to less classification imbalance. Experiments and discussion for threshold selection are given in the supplemental material (Kahng et al. 2023)

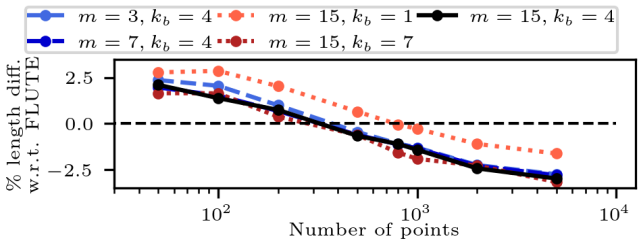

Figure 7: Dependence of NN-Steiner on k_b and m.

5 Conclusion

We present a mixed neural-algorithmic framework, NN-Steiner, for solving large-scale RSMT problems. This is the first neural architecture that has the capacity to solve RSMT approximately. In experiments, our framework shows generalization and scalability, and outperforms both heuristic algorithms and machine learning baselines on large instances.

Our ongoing research pursues the following directions. First, Arora's PTAS can solve higher-dimensional RSMT problems; this motivates us to generalize our framework to 3D IC designs. Second, the methodology behind NN-Steiner can be extended to compute obstacle-avoiding RSMTs, which again have important applications in VLSI design.

[3] We do not generate mixed-normal distributions of 5000 points because the $10^4 \times 10^4$ granularity is too restrictive.

Acknowledgments

This work is partially supported by NSF under grants CCF-2112665, CCF-2217058 and CCF-2310411, and by DARPA IDEA HR0011-18-2-0032. The authors thank Anastasios Sidiropoulos for helpful discussions during initial stages of this project. We also thank Qi Zhao for early explorations and inputs to the early writing of this paper.

References

Ajwani, G.; Chu, C. C. N.; and Mak, W.-K. 2010. FOARS: FLUTE based obstacle-avoiding rectilinear Steiner tree construction. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 30: 194–204.

Arora, S. 1998. Polynomial time approximation schemes for Euclidean traveling salesman and other geometric problems. *Journal of the ACM (JACM)*, 45(5): 753–782.

Bello, I.; Pham, H.; Le, Q. V.; Norouzi, M.; and Bengio, S. 2017. Neural combinatorial optimization with reinforcement learning. arXiv:1611.09940.

Bengio, Y.; Lodi, A.; and Prouvost, A. 2021. Machine learning for combinatorial optimization: a methodological tour d'horizon. *European Journal of Operational Research*, 290(2): 405–421.

Berman, P.; Fößmeier, U.; Karpinski, M.; Kaufmann, M.; and Zelikovsky, A. 1994. Approaching the 5/4—approximation for rectilinear Steiner trees. In *Proceedings of the European Symposium on Algorithms*, 60–71. Springer.

Chen, P.-Y.; Ke, B.-T.; Lee, T.-C.; Tsai, I.-C.; Kung, T.-W.; Lin, L.-Y.; Liu, E.-C.; Chang, Y.-C.; Li, Y.-L.; and Chao, M. C.-T. 2022. A reinforcement learning agent for obstacle-avoiding rectilinear Steiner tree construction. In *Proceedings of the International Symposium on Physical Design (ISPD)*, 107–115.

Chu, C.; and Wong, Y.-C. 2007. FLUTE: Fast lookup table based rectilinear Steiner minimal tree algorithm for VLSI design. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 27(1): 70–83.

Chu, C. C. N. 2004. FLUTE: fast lookup table based wirelength estimation technique. In *Proceedings of the IEEE/ACM International Conference on Computer Aided Design (ICCAD)*, 696–701.

Chu, C. C. N.; and Wong, Y.-C. 2005. Fast and accurate rectilinear Steiner minimal tree algorithm for VLSI design. In *Proceedings of the ACM International Symposium on Physical Design (ISPD)*, 28–35.

Cinel, S.; and Bazlamaçci, C. F. 2008. A distributed heuristic algorithm for the rectilinear Steiner minimal tree problem. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 27: 2083–2087.

Cygan, M.; Fomin, F.; Kowalik, L.; Lokshtanov, D.; Marx, D.; Pilipczuk, M.; Pilipczuk, M.; and Saurabh, S. 2015. *Parameterized Algorithms*. Springer.

Deudon, M.; Cournut, P.; Lacoste, A.; Adulyasak, Y.; and Rousseau, L.-M. 2018. Learning heuristics for the TSP by policy gradient. In *Proceedings of the International Conference on the Integration of Constraint Programming, Artificial Intelligence, and Operations Research (CPAIOR)*.

Fallin, A.; Kothari, A.; He, J.; Yanez, C.; Pingali, K.; Manohar, R.; and Burtscher, M. 2022. A simple, fast, and GPU-friendly Steiner-tree heuristic. In *Proceedings of the IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*, 838–847.

Garey, M. R.; and Johnson, D. S. 1977. The rectilinear Steiner tree problem is NP-complete. *SIAM Journal on Applied Mathematics*, 32(4): 826–834.

Garmendia, A. I.; Ceberio, J.; and Mendiburu, A. 2022. Neural Combinatorial Optimization: a New Player in the Field. *ArXiv*, abs/2205.01356.

Gasse, M.; Chételat, D.; Ferroni, N.; Charlin, L.; and Lodi, A. 2019. Exact combinatorial optimization with graph convolutional neural networks. *Advances in Neural Information Processing Systems*, 32: 15580–15592.

Gupta, P.; Gasse, M.; Khalil, E.; Mudigonda, P.; Lodi, A.; and Bengio, Y. 2020. Hybrid models for learning to branch. *Advances in Neural Information Processing Systems*, 33: 18087–18097.

Hanan, M. 1966. On Steiner's problem with rectilinear distance. *SIAM Journal on Applied Mathematics*, 14(2): 255–265.

Hu, Y.; Jing, T.; Feng, Z.; Hong, X.; Hu, X.; and Yan, G. 2006. ACO-Steiner: ant colony optimization based rectilinear Steiner minimal tree algorithm. *Journal of Computer Science and Technology*, 21: 147–152.

Hwang, F. K. 1976. On Steiner minimal trees with rectilinear distance. *SIAM Journal on Applied Mathematics*, 30(1): 104–114.

Juhl, D.; Warme, D. M.; Winter, P.; and Zachariasen, M. 2018. The GeoSteiner software package for computing Steiner trees in the plane: an updated computational study. *Mathematical Programming Computation*, 10(4): 487–532.

Kahng, A. B.; Lienig, J.; Markov, I. L.; and Hu, J. 2022. *VLSI physical design: from graph partitioning to timing closure*. Springer, second edition.

Kahng, A. B.; Măndoiu, I. I.; and Zelikovsky, A. Z. 2003. Highly scalable algorithms for rectilinear and octilinear Steiner trees. In *Proceedings of the Asia and South Pacific Design Automation Conference (ASPDAC)*, 827–833.

Kahng, A. B.; Nerem, R. R.; Wang, Y.; and Yang, C.-Y. 2023. NN-Steiner: A mixed neural-algorithmic approach for the rectilinear Steiner minimum tree problem. arXiv:2312.10589.

Khalil, E.; Dai, H.; Zhang, Y.; Dilkina, B.; and Song, L. 2017. Learning combinatorial optimization algorithms over graphs. *Advances in Neural Information Processing Systems*, 32: 6351–6361.

Kingma, D. P.; and Ba, J. 2014. Adam: a method for stochastic optimization. arXiv:1412.6980.

Li, Z.; Chen, Q.; and Koltun, V. 2018. Combinatorial optimization with graph convolutional networks and guided tree

search. *Advances in Neural Information Processing Systems*, 31: 537–546.

Liu, J.; Chen, G.; and Young, E. F. 2021. REST: Constructing rectilinear Steiner minimum tree via reinforcement learning. In *Proccedings of the 58th ACM/IEEE Design Automation Conference (DAC)*, 1135–1140.

Liu, S.; Zhang, Y.; Tang, K.; and Yao, X. 2022. How good is neural combinatorial optimization? A systematic evaluation on the traveling salesman problem. *IEEE Computational Intelligence Magazine*, 18: 14–28.

McCarty, E.; Zhao, Q.; Sidiropoulos, A.; and Wang, Y. 2021. NN-Baker: A neural-network infused algorithmic framework for optimization problems on geometric intersection graphs. *Advances in Neural Information Processing Systems*, 34.

Nair, V.; Bartunov, S.; Gimeno, F.; von Glehn, I.; Lichocki, P.; Lobov, I.; O'Donoghue, B.; Sonnerat, N.; Tjandraatmadja, C.; Wang, P.; et al. 2020. Solving mixed integer programs using neural networks. arXiv:1706.03762.

Prates, M.; Avelar, P. H.; Lemos, H.; Lamb, L. C.; and Vardi, M. Y. 2019. Learning to solve NP-complete problems: A graph neural network for decision TSP. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 4731–4738.

Sato, R.; Yamada, M.; and Kashima, H. 2019. Approximation ratios of graph neural networks for combinatorial problems. *Advances in Neural Information Processing Systems*, 32: 4081–4090.

Selsam, D.; Lamm, M.; Benedikt, B.; Liang, P.; de Moura, L.; Dill, D. L.; et al. 2018. Learning a SAT solver from single-bit supervision. In *Proceedings of the International Conference on Learning Representations (ICLR)*.

Vinyals, O.; Fortunato, M.; and Jaitly, N. 2015. Pointer networks. *Advances In Neural Information Processing Systems*, 28: 2692–2700.

Wang, Y.; Hong, X.; Jing, T.; Yang, Y.; Hu, X.; and Yan, G. 2005. The polygonal contraction heuristic for rectilinear Steiner tree construction. In *Proceedings of the Asia and South Pacific Design Automation Conference (ASPDAC)*, 1–6.

Wong, Y.-C.; and Chu, C. C. N. 2008. A scalable and accurate rectilinear Steiner minimal tree algorithm. In *Proceedings of the IEEE International Symposium on VLSI Design, Automation and Test (VLSI-DAT)*, 29–34.

Yehudai, G.; Fetaya, E.; Meirom, E. A.; Chechik, G.; and Maron, H. 2020. From local structures to size generalization in graph neural networks. In *Proceedings of the International Conference on Machine Learning*.